Ironic Specs

Release 0.0.1.dev764

OpenStack Ironic Team

Dec 05, 2024

CONTENTS

1 Themes 1

2 Specifications 59

3 Approved specifications 61

4 Back-log of ideas 167

5 Implemented specifications 193

6 Retired 1505

7 Indices and tables 1533

Bibliography 1535

CHAPTER

ONE

THEMES

During each PTG (before Pike - each design summit), we agree what the whole community wants to focus on for the upcoming release. This is the output of those discussions:

1.1 2025.1 Project Work Items

The latest virtual Project Team Gathering happened in October 2024. Ironic developers and operators discussed many different potential features and other ongoing work for the 2025.1 (Epoxy) release. These discussions are memorialized in this document, providing a list of the main priorities for the next development cycle. For more information please look at the link for each topic or contact the Ironic team on IRC.

Ironic contributors are busy; they work spanning multiple open source projects, and have varied downstream responsibilities. We cannot guarantee any or all planned work will be completed, nor is this a comprehensive list of everything Ironic may do in the next six months.

Each item in the table includes:

- Name of the work item, linked to the description
- **Category can be**
  - Maintenance: work that must be performed to keep Ironic working
  - Bugfix: work to enhance existing code to cover more corner cases and resolve bugs
  - Feature: a new Ironic feature that did not previously exist
- Champions are the people most familiar with the technologies involved, and are a good resource if youd like to implement the work item.
- Tracking link is a link to the bug (usually) tracking the work.

Table 1: 2025.1 Work Items

Name	Category	Tracking	Champions
Redfish Console Support	Feature	Redfish console support	TheJulia, JayF
Support OCI as a container for image files	Feature	Support OCI formatted URLs for artifact retrieval	TheJulia, help wanted
Support container-based deployment via bootc	Feature	Deployment of bootable containers	TheJulia
Networking: Project Mercury	Feature	Mailing list thread about new working group	BM Networking WG
Inspection hooks for Out-of-Band inspection	Feature	Standardize inspection hooks/data	cardoe
kea DHCP backend	Feature	Add a kea DHCP backend	cid
In-band Disk Encryption	Feature	Root device partition encryption (LUKS)	adam-metal3
Container-based IPA steps	Feature	Reserved step name format for agent container launch	JayF
Improve OEM handling in Sushy	Maintenance/Feature	Sushy to include OEM support	cardoe, dtantsur
Support NC-SI Hardware	Feature	Hardware that cannot be powered off	dtantsur
API response schema validation, OpenAPI Spec	Maintenance	Add OpenAPI support for Ironic via codegenerator	stephenfin, adamm-carthur5
Adopt pre-commit for linting in Ironic projects	Maintenance	N/A	JayF
Tinycore Alternative for IPA ramdisk	Maintenance	N/A	JayF
Retire ironic-lib	Maintenance	deprecate and retire ironic-lib	JayF, dtantsur

1.1.1 Goals Details

Redfish Console Support

Ironic has had a console interface with basic support for IPMI-based serial consoles for years. However, most machines now expose graphical KVMIP-based consoles via redfish interface. The Ironic team will

work on adding support for these consoles in the coming months.

Support OCI as a container for image files

OCI image urls (`oci://`) are getting more common in clouds, especially mixed clouds including both kubernetes and OpenStack. Ironic plans to implement support for fetching images from a container registry using this URL scheme alongside existing support for `http`, `https` and `glance`.

Support container-based deployment via bootc

Tooling has emerged in the cloud ecosystem allowing a container to be adapted for running on bare metal. We plan to implement one of these tools, `bootc`, as an additional deployment option over the coming months.

Networking: Project Mercury

Networking represents the next step for a truly standalone Ironic, and this means finding alternatives to Openstack-integrated scenarios and therefore to Neutron.

For complete usage in an enterprise use case, Ironic needs a means of networking control, which today is manual unless in a fully integrated OpenStack context. The Ironic team and interested operators have formed a working group around this and other networking improvements that can be made in Ironic as we refine our design for standalone networking.

There are no specific actions beyond further design discussion in the working group planned for this cycle.

Inspection hooks for Out-of-Band inspection

With the migration of inspection functionality into Ironic directly, we now have an opportunity to bridge some feature gaps between out-of-band (BMC) and in-band (agent) inspection methods. This feature will enable hooks to run based on out of band inspection data, similar to the existing support for in-band inspection data.

kea DHCP backend

Ironic currently has a single point of failure dependency in `dnsmasq`. The Ironic team will resolve this by adding support for an additional DHCP interface which will interface with `kea` DHCP server. This task is Ironic-specific; however the Ironic team also intends on specifying and implementing `kea` support in `neutron-dhcp-agent` as a next step.

In-band Disk Encryption

Linux servers frequently use disk encryption technology known as Linux Unifed Key Setup, (LUKS). The proposed implementation will allow operators to optionally enable data encryption at rest utilizing LUKS with a TPM-stored key.

Container-based IPA steps

Ironic Python Agent has long supported customization via HardwareManager, however, building and testing custom steps and HardwareManagers can be time consuming. With this change, the Ironic team will simplify agent customization by permitting operators to run steps out of containers, and modify them without being forced to rebuild their ramdisk.

Improve OEM handling in Sushy

Currently, sushy has minimal support for OEM endpoints, and historically required implementations of them e.g. `sushy-oem-drac` to remain out of tree. At the recent PTG, the Ironic team formed a consensus to adopt OEM logic directly into Sushy, simplifying our codebase and making it easier to support quirky vendor implementations of Redfish and potentially enabling utilization of OEM extensions.

Support NC-SI Hardware

Some hardware, such as that implementing the DTMF NC-SI specification, may not support power off. The Ironic team is working to support this hardware, and other hardware that may lack the ability to power off explicitly.

API response schema validation, OpenAPI Spec

The Ironic team is joining an effort by the OpenStack SDK team to generate OpenAPI specification documents from API code. This will ensure our API documentation will match the code by generating it from the code. As part of this work, the Ironic team will be refactoring handling of API microversions to help with the generation process and improve code readability.

Adopt pre-commit for linting in Ironic projects

Many OpenStack, and python projects in general, are adopting pre-commit to run the linting in their CI. The Ironic team is following this pattern, and will be consolidating lint jobs across all Ironic projects to be driven by pre-commit. This will also enable developers to enable a hook in their local git checkouts to have files automatically linted on save. We expect this to lower CI utilization by lessening the amount of lint failures on initial patch pushes and by consolidating multiple separate jobs e.g. `bandit`, `codespell`, and `hacking` all into a single test job.

Tinycore Alternative for IPA ramdisk

Tinycore has been the base for the Ironic Python Agent ramdisk (TinyIPA) used in the tests in the Ironic CI for a long time. Unfortunately it has become less and less tiny during the years, it lacks mirror https support, it uses a lightweight libc which caused issues multiple times, and we need to maintain a very specific series of scripts to be able to build it.

Wed like to explore alternatives to it, the main candidate being a gentoo based image that has also support in DiskImage-Builder.

Retire ironic-lib

Ironic-lib was originally created to enable sharing of deployment code between the now-obsolete `iscsi` driver and `direct` driver. With the `iscsi` driver removed, keeping the minimal shared code between IPA and Ironic is no longer worth the effort of managing an additional, separate project. The Ironic team will remove uses of the ironic-lib library, and we expect its final release in this cycle.

1.1.2 Release Schedule

Contributors are reminded of our scheduled releases when they are choosing items to work on.

The dates below are a guide; please view https://releases.openstack.org/epoxy/schedule.html for the full schedule relating to the release and https://docs.openstack.org/ironic/latest/contributor/releasing.html for Ironic specific release information.

Bugfix Release 1

The first bugfix release is scheduled to happen around the first week of December, 2024.

Bugfix release 2

The second bugfix release is scheduled to happen the first week of February, 2024.

Deadline Week

There are multiple deadlines/freezes in the final weeks of the release, please refer to the release schedule for exact dates.

Final 2025.1 (Integrated) Release

The final releases for Ironic projects in 2025.1 must be cut by March 24.

1.2 2024.2 Project Work Items

During the latest virtual Project Team Gathering happened between April 8 and 12, the Ironic developers and operators discussed multiple topics to plan the work for the next 2024.2 (Dalmatian) release. We summarize the outcome of the discussion in this document, providing a list of the main priorities for the next development cycle. For more information please look at the link for each topic or contact the Ironic team on IRC.

Ironic contributors are busy; they work spanning multiple open source projects, and have varied downstream responsibilities. We cannot guarantee any or all planned work will be completed.

Each item in the table includes:

- Name of the work item, linked to the description
- **Category can be**
  - Maintenance: work that must be performed to keep Ironic working
  - Bugfix: work to enhance existing code to cover more corner cases and resolve bugs
  - Feature: a new Ironic feature that did not previously exist
- Champions are the people most familiar with the technologies involved, and are a good resource if youd like to implement the work item.
- Tracking link is a link to the bug (usually) tracking the work.

Table 2: 2024.2 Work Items

Name	Category	Tracking	Champions
Ironic Documentation Improvements	Maintenance	N/A	JayF
API response schema validation	Feature	N/A	stephenfin
Merging Inspector into Ironic	Feature	Migrate inspection rules from inspector	masghar
Redfish Virtual Media Push / UpdateService	Feature	N/A	janders, dtantsur, TheJulia
Virtual Media TLS Validation	Feature	N/A	dtantsur
Slimming Down CI	Maintenance	N/A	team effort
Tinycore Alternative for IPA ramdisk	Maintenance	N/A	JayF, rpittau
Ironic Guest Metadata from nova	Feature	Ironic Guest Metadata	JayF
Service Steps templates	Feature	Expose templates for all steps, with project-awareness	JayF, TheJulia
Networking: Project Mercury	Feature	N/A	TheJulia
Ironic ARM CI	Feature	N/A	JayF, cid

1.2.1 Goals Details

Ironic Documentation Improvements

We will use the expertise of a professional docs writer to review all the Ironic documentation and propose a list of actionable work that can be performed to improve it.

We went through a first draft during the PTG, but we aim to finalize it and complete the main changes during the Dalmatian cycle.

API response schema validation

The SDK team would like to generate OpenAPI schemas for core OpenStack services and store them in-tree to ensure things are complete and up-to-date, to avoid landing another large deliverable on the SDK team, and to allow service teams to fix their own issues.

Eventually API documentation will switch from os-api-ref to a new tool developed and owned by the SDK team, but this is a stretch goal. When this happens, only the Sphinx extension itself will live out-of-tree (like os-api-ref today).

The list of advantages includes:

- Having a mechanism to avoid accidentally introducing API changes.

- API documentation will be (automatically) validated.
- Highlight bugs and issues with the API.

The first steps will be writing a spec and showing a framework example for one API.

Merging Inspector into Ironic

Ironic Inspector was originally created as a service external to Ironic. Now, its used by a large number of Ironic operators around the world and should be integrated with the primary service. This work has been progressing well. We will continue to work on this until it is complete.

Redfish Virtual Media Push / UpdateService

These are actually two separate proposals with a lot in common, the final goal being finding a way to facilitate virtual media booting and firmware updates by using a push model.

Well monitor the evolution of the Virtual Media Image Push proposal to the DMTF community, and well consider the UpdateService already present in the Redfish standard as a future alternative to be evaluated possibly in the next cycle.

Virtual Media TLS Validation

Fujitsu has already proposed the validation of the TLS connection to the BMC from Ironic, but we need to work on the other direction to validate the virtual media TLS connection from the BMC to the Ironic services.

Slimming Down CI

Ironic is one of the major CI consumers in terms of resources.

During the Caracal cycle weve been able to reduce the number of jobs run by the Ironic project, but weve also added some more. We came to the conclusion that we need to take an approach where we stack more tests in fewer jobs, trying to consolidate down jobs as much as possible and minimize boot interface variation testing.

During the Dalmatian cycle well work first on trying to have Redfish and ipmi in a single ironic job, and update the list of jobs to understand where we can avoid duplication.

Tinycore Alternative for IPA ramdisk

Tinycore has been the base for the Ironic Python Agent ramdisk (TinyIPA) used in the tests in the Ironic CI for a long time. Unfortunately it has become less and less tiny during the years, it lacks mirror https support, it uses a lightweight libc which caused issues multiple times, and we need to maintain a very specific series of scripts to be able to build it.

Wed like to explore alternatives to it, the main candidate being a gentoo based image that has also support in DiskImage-Builder.

Ironic Guest Metadata from Nova

We seek to unify the guest metadata sent to Ironic with that sent to libvirt.

Ironic currently only sets instance_info/instance_uuid, we want to expand this to include project_id, user_id and flavor name, such that we are more consistent with what is set in Libvirt guest metadata.

All of these fields are deleted when a node is undeployed, similar to instance_uuid today. The project_id might in the future be used to help with Ironic API RBAC.

Service Steps templates

We discussed this during the Caracal PTG in October, and as a result a spec was composed.

To move forward we need first to revise the spec with the latest outcome of the discussion during the most recent PTG.

Networking: Project Mercury

Networking represents the next step for a truly standalone Ironic, and this means finding alternatives to Openstack-integrated scenarios and therefore to Neutron.

For complete usage in an enterprise use case, Ironic needs a means of networking control, which today is manual unless in a fully integrated OpenStack context. Furthermore, the OpenStack integrated context has some known issues which makes it harder to adopt, so we plan to look for solutions to this difficult operations problem during this development cycle.

Ironic ARM CI

OpenStack Ironic uses extensive CI testing to validate things work.

While we support ARM, and have reports in the field of it working, we do not have any ARM representation, aside from unit tests, in our CI.

We aim to use ARM vms as we do for x86 vms and run one or more tempest scenario jobs in Ironic CI.

1.2.2 Release Schedule

Contributors are reminded of our scheduled releases when they are choosing items to work on.

The dates below are a guide; please view https://releases.openstack.org/dalmatian/schedule.html for the full schedule relating to the release and https://docs.openstack.org/ironic/latest/contributor/releasing.html for Ironic specific release information.

Bugfix Release 1

The first bugfix release is scheduled to happen around the first week of June, 2024.

Bugfix release 2

The second bugfix release is scheduled to happen the first week of August, 2024.

Deadline Week

There are multiple deadlines/freezes the final week of:

- Final release of client libraries must be performed
- Requirements freeze
- Soft string freeze - Ironic services are minimally translated; this generally doesnt apply to our services, such as API and Conductor, but may impact us via other projects which are translated.
- Feature Freeze - Ironic does not typically have a feature freeze, but we may be impacted by other projects that do have a feature freeze at this date.

Final 2024.2 (Integrated) Release

The final releases for Ironic projects in 2024.2 must be cut by September 27.

1.3 2024.1 Project Work Items

Ironic developers and operators met at the vPTG in October, 2023 to plan the 2024.1 (Caracal) release. This document is the output of that planning, and will not be updated once published. For information or current status of items in progress, please look at the linked bugs. For information about items completed, please see the Ironic release notes for the given release in question.

Ironic contributors are busy; they work spanning multiple open source projects, and have varied downstream responsibilities. We cannot guarantee any or all planned work will be completed.

Each item in the table includes:

- Name of the work item, linked to the description
- **Category can be**
  - Maintenance: work that must be performed to keep Ironic working
  - Bugfix: work to enhance existing code to cover more corner cases and resolve bugs
  - Feature: a new Ironic feature that did not previously exist
- Champions are the people most familiar with the technologies involved, and are a good resource if youd like to implement the work item.
- Tracking link is a link to the bug (usually) tracking the work.

Table 3: 2024.1 Work Items

Name	Category	Tracking	Champions
Eliminating Bug Backlog	Maintenance	2040552	JayF, dtantsur
Nova Ironic Driver sharding	Bugfix	ironic-shards	JayF, johnthetubaguy, TheJulia
Merging Inspector into Ironic	Maintenance	Migrate inspection rules from inspector	dtantsur
Develop plan for streamlined testing	Maintenance	N/A	TheJulia, iurygregory
Feature Parity with Metalsmith	Feature	2042575	dtantsur
Redfish HttpBoot	Feature	2032380	TheJulia
Step Templates	Feature	2027690	TheJulia, JayF
Cleanup Legacy Client Library Use in Ironic	Maintenance	2042493, 2042494, 2042495	stevebaker
Marking Multiple Drivers for Removal	Maintenance	N/A	JayF, TheJulia
Purging UEFI boot records	Feature	2042570, 2041901	dtantsur, stevebaker

1.3.1 Goals Details

Eliminating Bug Backlog

The Ironic core team has not prioritized bug triage or cleanup in recent years for a myriad of reasons. We will resolve this issue, and pay off related bug debt, during the 2023.2 cycle. A new bug deputy role will be created and rotate on a schedule. The deputy will be responsible for performing bug triage on new bugs, providing a report on bugs for the weekly meeting, and moderating at least one bug jam meeting a week to work through the backlog of neglected untriaged or outdated bugs. Our goal is that by the end of the 2023.2 cycle, no untriaged bugs older than two weeks will exist in any supported Ironic projects.

We define a triaged bug as one that has been read, prioritized, and usually commented on by an Ironic developer. Bug triage is important because it ensures that critical issues are seen quickly once reported.

Nova Ironic Driver Sharding

The failure scenarios around the existing Nova Ironic driver are grim: when an instance is provisioned, its permanently tied to the nova-compute that provisioned it and cannot be managed if that compute goes down. Additionally, at high scale, there are more race conditions due to the length of time it takes to query all Ironic nodes.

Instead, we will be adding support to Ironic, adding a sharding key which Nova can consume. This will allow us to split Ironic node management across a cluster of nova-compute services. Additionally, operators who want high availabililty will be able to setup active/passive failover on nova-compute services managing Ironic nodes.

This work was nearly completed last cycle, but was held to permit more automated testing to be completed before release.

Develop Plan for Streamlined Testing

Ironic is a complex system, with many different ways of configuring it. This leads to complicated matrices of test jobs. In the 2024.1 cycle, Ironic contributors will work to develop a plan to simplify the testing matrix, consolidating jobs where possible and identifying minimal next steps we can take next cycle.

Merging Inspector into Ironic

Ironic Inspector was originally created as a service external to Ironic. Now, its used by a large number of Ironic operators around the world and should be integrated with the primary service.

This work has been progressing well. We will continue to work on this until it is complete.

Feature Parity with Metalsmith

Metalsmith was created to simplify human interaction with the deployment of nodes using Ironic, primarily to support TripleO use cases. This simplified deployment model should be brought to all Ironic deployments. We will do this by adding a new deployment API to Ironic, allowing the deployment of a server through one single API call.

Additionally, we will add logic and usage ergonomics to Ironics interactive client, with the goal of provisioning a single machine with python-ironicclient with the new deployment API being as easy as deploying one today with metalsmith.

Once this work has completed, we will retire the dedicated metalsmith client.

Redfish HttpBoot

An often requested method utilized to boot machines is to leverage the ability to tell the remote BMC that the item to boot is available via an HTTP(S) URL.

By informing the BMC directly what to boot, we can remove fragile parts of the boot process such as TFTP or even in some cases PXE altogether. This will be integrated and share code with the existing implementations of virtual media on Redfish.

Step Templates

Ironic has used automated step processes to perform needed tasks on machines for a long time. However, these have always operated in two modes, either fully automated, or fully specified. This makes it difficult to use these step-based automations in a way that is friendly to a less-technical operator.

To address this, we propose changing the current concept of Deploy Templates to a generic concept of Step Templates. These templates would be RBAC-aware, and would be able to be used in the API anywhere youd currently pass a dictionary of steps, such as for manual cleaning or service.

Our goal for the this cycle will be to fully specify this feature and identify any potential pain points that could occur during the upgrade process.

Cleanup Legacy Client Library Use in Ironic

Ironic is integrated with a number of OpenStack services, interacting via client requests. Requests to some services (nova, neutron, keystone) are made via openstacksdk while others (swift, cinder, glance) are still made via the legacy client libraries. During the caracal cycle, we intend to complete the migration to use only openstacksdk.

Marking Multiple Drivers for Removal

Ironic has a long history of working closely with hardware vendors, and theyve reciprocated in kind by helping develop drivers and manage CI against them for many years. However, with the emergence of advanced hardware management standards, like Redfish, the need for vendor-specific drivers is diminishing.

Furthermore, many of the legacy drivers originally targeted at older, nonstandard interfaces or server management solutions, due to their naming, often sound like they more closely match a given vendors hardware than the Redfish driver. These drivers, while still useful for older hardware which may not support redfish, are not ideal for modern hardware.

For this reason, and others, we have selected a number of drivers to be marked for removal. We do not intend to actively remove any of these drivers until it is clear any hardware they exclusively support has gone end of life. We are primarily taking this action to indicate to operators that they should be provisioning new hardware with Redfish-based drivers.

The Ironic community is extremely grateful to these vendors for supporting their drivers in Ironic for so long, and for supporting the open standards which, in most cases, are obsoleting the need for specific drivers.

Please note that suggested alternatives are not tested by Ironic CI and are suggestions based on specified system support or vendor recommendations.

Table 4: Drivers to be marked for removal

Driver	Hardware	Alternatives
`ibmc`	Huawei	`redfish`
`idrac-wsman`	Dell (iDRAC 5,6)	`idrac-redfish` (iDRAC 7+)
`xclarity`	Lenovo (cluster manager)	`redfish` (to individual systems)
`ilo`	HPE servers (iLO 5 or older)	`redfish` (iLO 6 and newer)

Purging UEFI boot records

Stale EFI boot records can cause problems with booting or adding new records. Since Ironic manages a node it should be responsible for removing any existing boot record which resembles a disk or attached boot device (USB, CDROM, etc).

This can be done in-band with a step which can optionally be included during cleaning or deployment. However it also needs to be possible to do it via Redfish to handle the cases where the node wont boot at all due to stale incorrect records.

1.3.2 Release Schedule

Contributors are reminded of our scheduled releases when they are choosing items to work on.

The dates below are a guide; please view https://releases.openstack.org/caracal/schedule.html for the full schedule relating to the release and https://docs.openstack.org/ironic/latest/contributor/releasing.html for Ironic specific release information.

Bugfix Release 1

The first bugfix release is scheduled to happen around the first week of December, 2023.

Bugfix release 2

The second bugfix release is scheduled to happen the first week of February, 2024.

Deadline Week

There are multiple deadlines/freezes the final week of February: * Final release of client libraries must be performed * Requirements freeze * Soft string freeze - Ironic services are minimally translated; this generally doesnt apply to our services, such as API and Conductor, but may impact us via other projects which are translated. * Feature Freeze - Ironic does not typically have a feature freeze, but we may be impacted by other projects that do have a feature freeze at this date.

Final 2024.1 (Integrated) Release

The final releases for Ironic projects in 2024.1 must be cut by March 25.

1.4 2023.2 Project Work Items

Most Ironic contributors have a large number of responsibilities, including but not limited to, keeping CI working across all branches and projects, backporting bugfixes, and any downstream job responsibilities that do not include Ironic contribution.

Because of how much this work can vary over a six month release cycle, we do not specify timelines for specific work items. Instead, we document planned items we would like to see completed, and then allow the community to pick up and work on them as time permits. Items are listed in no particular order.

This document represents our outlook for 2023.2, and will not be updated once published. For information or current status of items in progress, please see the Ironic Whiteboard at https://etherpad.openstack.org/p/IronicWhiteBoard. For information about items completed, please see the Ironic release notes.

Each item in the table includes:

- Name of the work item, linked to the description
- **Category can be**
  - Maintenance: work that must be performed to keep Ironic working
  - Bugfix: work to enhance existing code to cover more corner cases and resolve bugs
  - Feature: a new Ironic feature that did not previously exist
- Champions are the people most familiar with the technologies involved, and are a good resource if youd like to implement the work item.

Table 5: 2023.2 Work Items

Name	Category	Champions
Nova Ironic Driver sharding	Bugfix	JayF, johnthetubaguy, TheJulia
Remove default use of MD5	Bugfix	TheJulia
Merging Inspector into Ironic	Maintenance	dtantsur
Service Steps	Feature	TheJulia
Conductor Graceful Shutdown	Bugfix	stevebaker
FIPS Compatibility jobs in CI	Feature	Ade Lee
Cross-conductor communication	Feature	TheJulia
Hierarchical Nodes	Feature	TheJulia
Improving Deploy Kernel/Ramdisk Config	Feature	None
IPA Communication	Bugfix	kaloyank
Firmware Updates	Feature	iurygregory, dtantsur, janders

1.4.1 Goals Details

Nova Ironic Driver Sharding

The failure scenarios around the existing Nova Ironic driver are grim: when an instance is provisioned, its permanently tied to the nova-compute that provisioned it and cannot be managed if that compute goes down. Additionally, at high scale, there are more race conditions due to the length of time it takes to query all Ironic nodes.

Last cycle we added support for Ironic, to allow assigning a shard key to nodes that can be used by clients, including Nova, can consume to split Ironic node management across a cluster of services.

We hope to continue progress on this goal by implementing support for sharding APIs in openstacksdk and python-ironicclient. Then, we will add support in the Nova driver and networking-baremetal for sharding queries.

Remove default use of MD5

The MD5 hashing algorithm is still supported in Ironic for image hashing. This is not ideal as MD5 is broken. This work will be a breaking change; forbidding use of MD5 hashes by default. Operators who wish to continue using MD5 for API compatibility reasons will be able to re-enable it via config.

Merging Inspector into Ironic

Ironic Inspector was originally created as a service external to Ironic. Now, its used by a large number of Ironic operators around the world and should be integrated with the primary service.

Last cycle, the Node Inventory API was implemented in Ironic. Next, we will move the rest of inspector functionality into Ironic. For more information, see the relevant specs:

Merge Inspector into Ironic Migrate inspection rules from Inspector

Service Steps

Ironic uses steps to perform actions on a node during deployment or cleaning. Wed like to extend this concept of steps to allow for maintenance on actively deployed nodes. This new Service Steps (formerly referred to as Active Steps) feature will allow operators to perform a firmware update or any other automated action on a provisioned, ACTIVE node.

We will also be implementing some basic flow control steps into Ironic. These commands, such as hold and pause will enable using steps to inform Ironic to wait for an external API client or a configured period of time before continuing. Additionally, we will evaluate more hardware and API actions to expose as steps, such as power and BMC actions.

Conductor Graceful Shutdown

Initial work on gracefully shutting down by waiting for all locks to be released was completed early in the Bobcat cycle. Next, we will work on support for draining a conductor of tasks before shutting it down. The goal is to ensure no in-progress actions are interrupted. This will allow for truly non-disruptive conductor shutdowns and restarts.

FIPS Compatibility jobs in CI

FIPS compatibility is a cross-project goal in OpenStack. We hope to have CI jobs added to identify areas in Ironic that are not FIPS compatible. No major incompatibilities are anticipated, but we may need to update some hashlib.md5() calls and other minor changes.

Cross-conductor communication

Many conductor management actions taken on a node are written with assuming a single conductor will perform them. This is not great for availability or maintenance scenarios. We will be looking to implement some form of cross- conductor communication to permit conductors to hand off work when being shut off.

For more information, see the cross-conductor rpc hand-off spec.

Hierarchical Nodes

Many new pieces of technology, such as DPUs, are presenting more complex interfaces to hardware integrators. Architectures have emerged with nested devices, with multiple firmwares and multiple nested operating systems to manage. In order to support these, we are introducing parent/child relationship to nodes. For more information, see the DPU management/orchestration spec.

Improving Deploy Kernel/Ramdisk Config

Ironic currently offers two places to easily manage deploy kernel and ramdisk: configuration file, for global settings, and node metadata for per-node overrides. This presents a problem for operators who want to operate Ironic with hardware that requires different ramdisks; such as ARM and x86 they will have to make N API calls for N nodes to update their non-default arch ramdisks.

To resolve this problem, well be introducing config to allow setting default ramdisks per-architecture. This will allow operators to set a different default ramdisk for ARM and x86 nodes.

IPA Communication

The current method of communication between Ironic and the Ironic Python Agent ramdisk, including the agent token for security, is fragile in some use cases, including neutron-integrated deployments with fast-track mode enabled.

Ironic contributors will be looking at ways to improve the communication with the goal in mind to improve behavior around complex scenarios like the one mentioned above.

For more information, see the IPA communication spec.

Firmware Updates

Ironic currently supports firmware updates via steps run in cleaning or deployment. However, this is not ideal because it requires significant operator understanding to perform updates.

Instead, as we have for BIOS and RAID, we will create a dedicated firmware update interface, which will give a standard way to upgrade and manage firmware.

See the firmware update spec for more information.

1.4.2 Release Schedule

Contributors are reminded of our scheduled releases when they are choosing items to work on.

The dates below are a guide; please view https://releases.openstack.org/bobcat/schedule.html for the full schedule relating to the release and https://docs.openstack.org/ironic/latest/contributor/releasing.html for Ironic specific release information. Please reach out to the Ironic team if you would like to request a bugfix release.

Bugfix Release 1

The first bugfix release opportunity is the first week of May.

Bugfix release 2

The second bugfix release opportunity is the first week of July.

Deadline Week

There are multiple deadlines/freezes the week of August 28th: * Final release of client libraries must be performed * Requirements freeze * Soft string freeze - Ironic services are minimally translated; this generally doesnt apply to our services, such as API and Conductor, but may impact us via other projects which are translated. * Feature Freeze - Ironic does not typically have a feature freeze, but we may be impacted by other projects that do have a feature freeze at this date.

Final 2023.2 (Integrated) Release

The final releases for Ironic projects in 2023.1 must be cut by September 29, 2023.

1.5 2023.1 Project Work Items

Most Ironic contributors have a large number of responsibilities, including but not limited to, keeping CI working across all branches and projects, backporting bugfixes, and any downstream job responsibilities that do not include Ironic contribution.

Because of how much this work can vary over a six month release cycle, weve chosen to no longer specify timelines for specific work items. Instead, we will document work items we would like to see completed, and then allow the community to pick up and work on them as time permits.

We strongly encourage all corporate and individual contributors to dedicate resources to help maintain the commons; including CI and stable branches. The more contributors we have helping keep the commons healthy, the more time we have to work on new items.

This document represents our outlook for 2023.1, and will not be updated once published. For information or current status of items in progress, please see the Ironic Whiteboard at https://etherpad.openstack.org/p/IronicWhiteBoard. For information about items completed, please see the Ironic release notes.

Each item in the table includes:

- Name of the work item, linked to the description
- **Category can be**
  - Maintenance: work that must be performed to keep Ironic working
  - Bugfix: work to enhance existing code to cover more corner cases and resolve bugs
  - Feature: a new Ironic feature that did not previously exist
- Champions are the people most familiar with the technologies involved, and are a good resource if youd like to implement the work item.

Table 6: 2023.1 Work Items

Name	Category	Champions
SQLAlchemy 2.0 Compatibility	Maintenance	TheJulia
Nova Ironic Driver sharding	Bugfix	JayF, johnthetubaguy, TheJulia
Cleaning up RAID created by tenants	Bugfix	dtantsur, ftarasenko
Remove default use of MD5	Bugfix	TheJulia
Merging Inspector into Ironic	Maintenance	dtantsur, jjelinek
Active Steps	Feature	moshele, janders
Conductor Scaling & Locking	Bugfix	stevebaker

1.5.1 Goals Details

SQLAlchemy 2.0 Compatibility

Our DB layer is currently using SQLAlchemy 1.4 or older, and relies heavily on autocommit behaviors. Additionally as part of the migration to 2.0, SQLAlchemy now requires developers be more explicit about querying than in previous versions. There is a significant amount of work to be done as part of this migration, but the resulting DB layer in the code should be clearer as a result.

Nova Ironic Driver Sharding

The failure scenarios around the existing Nova Ironic driver are grim: when an instance is provisioned, its permanently tied to the nova-compute that provisioned it and cannot be managed if that compute goes down. Additionally, at high scale, there are more race conditions due to the length of time it takes to query all Ironic nodes.

Instead, we will be adding support to Ironic, adding a sharding key which Nova can consume. This will allow us to split Ironic node management across a cluster of nova-compute services. Additionally, operators who want high availabililty will be able to setup active/passive failover on nova-compute services managing Ironic nodes.

Cleaning up RAID created by tenants

It has come to the Ironic communitys attention that more and more cases are arising where customers of BMaaS systems are doing things such as setting up their own RAID sets. This can complicate undeploy/cleaning, and in some cases redeployment of the machine.

To address this, the Ironic community is considering changes to the overall cleaning workflow to disassemble discovered raid sets. While this is not yet clearly defined, we hope that doing so will improve operators and end users experiences.

Remove default use of MD5

The MD5 hashing algorithm is still supported in Ironic for image hashing. This is not ideal as MD5 is broken. This work will be a breaking change; forbidding use of MD5 hashes by default. Operators who wish to continue using MD5 for API compatibility reasons will be able to re-enable it via config.

Merging Inspector into Ironic

Ironic Inspector was originally created as a service external to Ironic. Now, its used by a large number of Ironic operators around the world and should be integrated with the primary service. Now is a good time to do this work as well, as Ironic Inspector also needs to be updated to work with SQLAlchemy 2.0.

Active Steps

Ironic uses steps to perform actions on a node during deployment or cleaning. Wed like to extend this concept of steps to allow for maintenance on actively deployed nodes. This new Active Steps feature will allow operators to perform a firmware update or any other automated action on a provisioned, ACTIVE node.

Conductor Scaling & Locking

Traditionally, Ironic has taken an aggressive approach to locking nodes while work is being performed at the expense of operations failing when a conductor is shutdown or unavailable. This change will allow operators to gracefully shutdown a conductor from the cluster, ensuring that no in-progress actions will fail. As part of implementing and improving this, we will also need to narrow the situations in which Ironic will lock a node and improve our methods of locking.

1.5.2 Release Schedule

Contributors are reminded of our scheduled releases when they are choosing items to work on.

The dates below are a guide; please view https://releases.openstack.org/antelope/schedule.html for the full schedule relating to the release and https://docs.openstack.org/ironic/latest/contributor/releasing.html for Ironic specific release information.

Bugfix Release 1

The first bugfix release is scheduled to happen the first week of December.

Bugfix release 2

The second bugfix release is scheduled to happen the first week of February.

Deadline Week

There are multiple deadlines/freezes the week of February 13th: * Final release of client libraries must be performed * Requirements freeze * Soft string freeze - Ironic services are minimally translated; this generally doesnt apply to our services, such as API and Conductor, but may impact us via other projects which are translated. * Feature Freeze - Ironic does not typically have a feature freeze, but we may be impacted by other projects that do have a feature freeze at this date.

Final 2023.1 (Integrated) Release

The final releases for Ironic projects in 2023.1 must be cut by March 17th, 2023.

1.6 Zed Project Themes

1.6.1 Themes

Table 7: Zed Themes

Theme	Primary Contacts	Target
CI Health	iurygregory, rpittau, TheJulia	1,2,3
Ironic Safeguards	TheJulia, arne_wiebalck	2,3
RBAC Phase 2	TheJulia	2,3
OpenConfig support in net-baremetal	hjensas	2,3

Schedule Structure

Zed Release Schedule.

Sprint 1

The release for this sprint will happen on May (02-06)

Sprint 2

The second release is scheduled to happen on Aug 15 - Aug 19

Sprint 3

This is the release that will create the stable/zed branch, according to the release team schedule we have:

- non-client libraries: Aug 22 - Aug 26 (R-6)
- client libraries: Aug 29 - Sep 02 (R-5)
- Final RCs and intermediary releases: Sep 26 - Sep 30 (R-1)

1.6.2 Goals Details

CI Health

During the PTG we notice that we need to enable CI testing for some features that were included in other releases, and also enable upgrade testing following the new release cadence adjustment.

Ironic Safeguards

We will introduce the ability to safeguard ironic deployments when running cleaning operations. It will be possible to limit the maximum number of concurrent cleaning operations for nodes in the infrastructure and also be able to specify a list of disks that should/shouldnt be cleaned for each node.

RBAC Phase 2

Following the TC goal Consistent and Secure RBAC we will be working on the Phase 2 described during this cycle.

OpenConfig support in net-baremetal

We will be adding device configuration capabilities for networking-baremetal, since in multi-tenant BMaaS there is a need to configure the ToR network devices (Access/Edge Switches) and many vendors have abandoned their ML2 mechanism plug-ins to support this.

1.7 Yoga Project Themes

1.7.1 Themes

Table 8: Yoga Themes

Theme	Primary Contacts	Target
ARM effort	rpittau	1
Redfish improvements	dtantsur	2
Nova improvements	TheJulia	3
Attestation Interface	sdanni, lmcgann, TheJulia, iurygregory	2
Enhancing storage cleaning	janders	3
Start Merging Ironic Inspector in Ironic	tkot, dtantsur	3
Drop privileged operations from Ironic	iurygregory, dtantsur, rpittau	2
Make it easier to deploy and operate	dtantsur, TheJulia	3
Tempest on bifrost	iurygregory	2
Troubleshooting FAQ/Guide	Ironic contributors	3
RBAC on ironic-tempest	TheJulia	3

Schedule Structure

Sprint 1

The release for this sprint will happen on the first week of December (06 - 10).

Sprint 2

The second release is scheduled to happen on the first week of February (01-04).

Sprint 3

This is the release that will create the stable/yoga branch, according to the release team schedule we have:

- non-client libraries: Feb 14 / Feb 18.
- client libraries: Feb 22 / Feb 25.
- final release: Mar 21 / Mar 25.

1.7.2 Goals Details

ARM effort

The interest in ARM Hardware has grown, since the opendev infra has some resources we will start building ramdisk image for this architecture. We will have images published for the architecture and having bifrost testing.

Attestation Interface

Recent interest in having an integration with Keylime has brought forth interest in resurrecting the attestation interface which was proposed some time ago to provide an integration point for Ironic to have the understanding and capability to take the appropriate action in the event a machine has been identified to no longer match the expected profile.

Enhancing storage cleaning

We want to improve storage cleaning in hybrid scenarios, the proposal is described in Improve efficiency of storage cleaning in hybrid NVMe+other storage configurations.

Tempest on bifrost

The idea here is that we can improve bifrost so we can run tempest, having this can reduce the dependency on devstack in our CI and also for 3rd Party CI.

Start Merging Ironic Inspector in Ironic

Based on the PTG discussions, we will provide a new home for introspection rules using a new format (still need to be discussed with the community), we also want to add the ability to generate ironic-inspector iPXE scripts.

Troubleshooting FAQ/Guide

We should always engage in trying to improve the user experience, this is something that we as a community should improve.

RBAC on ironic-tempest

Weve reached consensus we want to add an additional set of tests in an attempt to help provide additional guards in the terms of this should never work. The prime purpose of such is to help the community and operators identify major issues and potential configuration issues and have comprehensive and exhaustive testing.

Redfish improvements

Refactor sushy to add features/deprecations for newer Redfish standards. Some improvements already includes: switching constants to python enum, auto-generation of code for enums.

Nova improvements

With twenty percent of OpenStack compute deployments leveraging Ironic as their hypervisor, it is critical for the ironic community to take the needs and issues experienced by those operators critical in the interaction between Nova and Ironic. Most of the issues revolve around attempting to fit a model of bare metal into a model of virtual machines. Obviously, this has issues, but we will be spending some bandwidth to improve the overall experience in an attempt to make things better.

Drop privileged operations from Ironic

Given the memory impact that the *oslo.privsep* could cause we decided to drop privileged operations from Ironic, the work will be trackted in the Story 2009704.

Make it easier to deploy and operate

It will consist of improvements that aim to make the operator life easier, like: removing the need for some manual commands during installation, automatic movement of machines through the workflow.

1.8 Xena Project Themes

Alala[1]

During the Wallaby development cycle, we had a goal in which History favors the bold. It was clear early on in that development cycle that we just had too much work coming up for contributors, that it simply was not going to be gotten to during the cycle. Unfortunately, it happens and in the grand scheme of things it is good that the community recognized it simply did not have the capacity to push it forward.

It is hard lesson to learn. But an important lesson.

The theme of that specific work is still true. History does favor the bold. And while we shouldnt have a war cry, much less ever declare war. We should be tactical in our thinking, and execution to meet everyones needs. Be it operator, contributor, or ultimately end user.

And so, it seems the best path is to be bold. For us to charge forth! Declare our vision, and spread the song of our successes. Of course, needs change. Requirements change. But underlying themes should be remembered.

1.8.1 Purpose

Every deployer and developer has different wants, desires, needs, and hopes. The intention of this document is to lay out a consensus of the community as to what work we feel is important for the release. A work item noted in this document will get more reviews and support from the community than new or unprioritized work, but theres no guarantee theyll get done in the Xena cycle, or at all.

This is a list of goals the Ironic team is prioritizing for the Xena development cycle, in order of relative size with context of our dependencies and roughly referenced against the anticipated sprints and release cycle for the Xena development cycle.

The primary contact(s) listed are responsible for tracking the status of that work and herding cats to help get that work done. They are not the only contributor(s) to this work, and not necessarily doing most of the coding! They are expected to be available on IRC and the ML for questions, and report status on the whiteboard for the weekly IRC sync-up. The number of primary contacts is typically limited to 2-3 individuals to simplify communication. We expect at least one of them to have core privileges to simplify getting changes in.

Note

In the interests of keeping our work fun and enjoyable, while continuing to foster community engagement, this document may have a bit of silliness intertwined. It is all okay, we havent lost all of our sanity, yet.

[1] Alala is a reference to Greek mythology where it was the female personification of raising a war cry. In this context, it is a reference to the television show Xena: Warrior Princess.

1.8.2 Shifting to a Theme

The Ironic community reached consensus during the Xena PTG that our choice of word to describe things needed to be revised. Specifically, one persons priority is not another persons priority, and priorities can shift rapidly for individual contributors based upon present needs of their employers.

And so we seemed to reach consensus that the word was more along the line of `themes` instead of `priorities`. We can have a higher level theme for the release or cycle but then we can have some specific smaller themes which may, or may not make it into the release. From the outside, this may seem like a drastic change, but this is more in alignment with existing project practice and the realities we live with.

Going back to the *Purpose* of this document, what we document here is our consensus forecast. And is subject to change. Consensus may shift after the fact, but it is of the utmost importance to have a mutually agreeable starting point, which this document provides.

1.8.3 Goals

Priority	Primary Contacts	Target
Being Bold	Ironic Developers, Baremetal SIG	Theme
Finishing anaconda deployment	zer0cool, rpittau	Sprint 1
iSCSI deployment removal	dtantsur	Sprint 2
Database performance	TheJulia	Sprint 2
Node error history	kaifeng, arne_wiebalck	Sprint 2
Enhancing storage cleaning	janders	Sprint 2
Move to oslo.privsep	iurygregory, rpittau	Sprint 2
Virtual Media visibility	iurygregory, dtantsur	Sprint 2
Driver structure and model	TheJulia, TheJulia	Sprint 3
Finishing Secure RBAC	TheJulia	Sprint 3
Snapshots	kaifeng, TheJulia	Future
Security Interface	kaifeng, ljmcgann, rpittau, sdanni	Future
Keylime	ljmcgann, sdanni	Future
Boot from URL	Multiple contributors	Future

Schedule Structure

The indicator for this schedule is to help provide those reviewing this document a rough idea of when one may anticipate functionality to merge and be released. Things may merge sooner or later.

Sprint 1

We anticipate the release from the first sprint to be on the week of May 31st, 2021. This is six weeks after the initial planning week.

Sprint 2

The second sprint is anticipated to start the week of June 7th, 2021 and proceed until July 19th, 2021. This is approximately 7 weeks for the second sprint.

Sprint 3

After the second sprint release, The third sprint is anticipated to end the week of September 6th. Features and Driver enhancements which are not review ready will not be considered to hold the release on. This is anticipated to be a window of seven weeks. In essence, this can be viewed as a freeze but is more of a the release train must depart on time model.

Anticipated release date is September 9th.

Post Release Sprint

After the release, it seems to be a good time to ensure weve taken of needful and any necessary backports, along with bug triaging, and resolution of any low-hanging-fruit bugs with major impact. Example being documentation not rendering configuration options, or deployments fail under Y conditions.

This time also aligns with the general community shift after releases, where people tend to take some time off to recharge before planning, and the planning steps and discussions take place.

This sprint is anticipated to end October 8th, 2021. Additional releases may take place to address bug fixes that need to be backported during this window. The following week is anticipated to be the Project Teams Gathering and the start of the following development cycle.

Theme

General thematic work for general improvement in an area fall under the classification of theme. Largely this is work that may run the course of an entire release cycle or longer, where small incremental improvements or related work takes place.

Future

Items in the future which we as a community do not have a firm idea of *when* this may merge. Being on this list does express that interest exists in the community to push this effort forward during the cycle.

1.8.4 Goals Details

In no particular order

Being Bold

In alignment with our general theme of alala, it is important for us to be bold, and acknowledge when we do and do not have capacity to push things forward. At the same time, we need to broadcast out. We need to speak of our successes. Our wins and failures. And everything in between.

This visibility, can unfortunately make some of us uncomfortable, but it takes many forms. Mentorship, Public Speaking, and engagement outside of your day to day primary mission focus. This is also how we grow. How we grow ourselves, along with the community.

So be bold, think alala and go forth and cross the division. Do the thing which is uncomfortable. Propose the crazy idea. And when the times permit it, get on stage at a local conference, and speak of the experience.

For history favors the bold.

Of course, we also want to keep our santiy, or at least some of it.

Database Performance

During the Xena development cycle, the community driven Secure RBAC work added additional database interactions with-in the API which has the general effect of increasing load upon the database when `project` scoped tokens request items. As this is the new direction for access control in OpenStack, we need to make sure we are not making this a burden for operators with API access activity. Since some configuration patterns, could result in even more activity, depending on how the end operator chooses to configure their deployment.

Also during the cycle, one of the larger operators encountered a thundering herd situation. In essence, the database could not keep up. We need to try and be smart to prevent some of these situations from happening, or at least minimize the impact as many operators now also launch services using Kubernetes, which can result in all services coming online at the same moment, an aspect which aggravates a thundering herd.

It should be noted, not all of this is intended to be feature work, as some of the work product will end up being backported to the Wallaby release which may take slightly different approaches.

This work may also extend into APIs for bulk data, but this ultimately also requires additional information before we can make such a decision.

Finishing Secure RBAC

The community driven Secure RBAC work which has really been underway community wide for a number of years made a large push, also community wide for projects to implement and adopt new policies whilst deprecating their old policies.

We anticipate we will seek to remove the old policies during the Xena cycle, however we need to consider *Database Performance* and needs of the operator.

Secure RBAC is also a fairly new configuration, we may find cross-service bugs or issues, that require additional work. This ultimately was somewhat expected.

Node error history

To boldly go forth, we must provide more insight into error history of nodes. The concept of adding support to record the important events and surface them in a human parsable way has long been under discussion and been a desired feature. It is time we make it happen.

This work was started during the Wallaby development cycle but we did not have the capacity to move it forward last cycle. This cycle we ought to finish it.

Finishing anaconda deployment

Some operators are invested in Anaconda configurations and using Anaconda kickstart files to facilitate deployments. More information can be found in anaconda deployment specification.

This work started during the Wallaby development cycle and continues. It should be wrapped up early in the first sprint as a dependency was identified too late in the Wallaby development cycle to be addressed until Xena.

Snapshots

A major compatibility gap with Novas Compute interaction with VMs that is lacking with Ironic baremetal nodes is support for Snapshots. This is a bit of a complex problem which may require an iterative development process. This is presently under discussion and the community is interested in the functionality. Information about this feature can be found in the snapshot specification document.

Move to oslo.privsep

This effort is being carried over from the prior cycle as it became clear the work required would take longer than time existed for us to move the changes forward. More information can be found in the migrate to privsep goal documentation.

Security Interface

Recent interest in having an integration with Keylime has brought forth interest in resurrecting the security interface which was proposed some time ago to provide an integration point for Ironic to have the understanding and capability to take the appropriate action in the event a machine has been identified to no longer match the expected profile.

Keylime

The Keylime project is an open source system security and attestation framework which was originally developed at MIT Lincoln Laboratory and has evolved in close contact with the Ironic community over the past few years while it has become a project on its own.

Keylime helps ensure that the underlying hardware of a deployment has not been tampered with, and that ultimately the system is running the firmware and software expected. It does this with low level Trusted Platform Module integration, and a set of services that an operator may choose to deploy.

Ultimately having support for integration helps ensure a greater level of operational security by helping operators identify and isolate machines which have had malicious actions taken on them and also potentially help increase the level of security of the deployment process by helping identify if a malicious actor has attempted to modify a running ramdisks contents.

This work requires the implementation of the Security Interface.

Boot from URL

This is a long sought after feature, and one more likely to surface as time goes on. Part of the conundrum is the multiple routes possible in what is interpreted as Boot from URL. Luckily Redfish has defined a standard interface to assert the configuration via the BMC.

At a minimum this cycle, we would like to make a step forward in attempting to support this functionality such that we can support it when vendors implement the feature outside of vendor OEM specific mechanisms.

Basic information on the hope of this can be found at HTTPClient Booting. Additional prior art can be found in the `ilo` hardware type as well, but the hope is to support this generically, if at all possible.

Virtual Media Visibility

One of the biggest headaches for the operator and developer community, when it comes to virtual media, is the nature of the integration point in firmware.

This feature set involves a complex interaction of open source software with semi-proprietary or standards based APIs over an HTTP connection. Often this is greatly complicated by the teams which develop the firmware often are on entirely separate teams inside organizations which doesnt have the level of insight that the community has. Ultimately, the result is sometimes virtual media breaks.

The idea is simple. Identify if the machine is *known-good* for virtual media and expose that in some way/shape/form, if appropriate, along with what is historically treated as tribal knowledge in terms of workarounds or potential fixes. This may be contentiouns to some, because perceptions do matter, but so does usability and we need to somehow balance this ever evolving pain point.

Driver structure and model

Our driver model has the advantage of operators being able to be very specific and ultimately have some level of knowledge or trust in the behavior of the node. Except, that power also comes with sources of confusion, and some pain points which are related where some overlapping code *can* result in unintended consequences.

We recognize the need to try and build consensus on one or more improvements to help alleviate some of these issues and possibly provide a forward path, while still providing a level of flexibility.

This is largely only anticipated to be a specification document this cycle, which may only be used to settle on consensus for policy moving forward. This may also drive future code enhancements, but we wont know until consensus is reached on this topic.

Related to part of this is story 2008804 and story 2005328 which propose some ideas related to this.

Enhancing storage cleaning

Storage is a complex issue with Bare Metal. In essence two different schools of thought exist which support operators. The first is where we want to absolutely make sure nothing is still present anywhere on the machine. Some operators need this level of cleanliness. Where as others just need to know they can safely re-deploy on to the machine without repercussions.

Also, as time shifts, so do our positions and takes, so we want to make metadata wipe more akin to help provide a greater level of assurance to the just want to be able to reuse my machine safely group of operators.

This may result in some changes to how Secure Erase/Format operations are handled, as well as additional portions of data to be removed from disks to aid in reuse. Specifically for operators with Ceph.

iSCSI deployment removal

The first deployment method in Ironic, is also one of the more we just need to trust the underlying mechanisms and hope nothing happens sort of drivers. It turns out those substrates dont handle intermittent transient failures or issues such as a port state resetting mid-flight. Due to this, deployments are easily broken or interrupted which is not ideal in varying infrastructures with different network configurations. These factors led the community to reach consensus that it was time to deprecate this deployment mechanism, and ultimately remove it from Ironic.

We anticipate it to disappear before our final release for the Xena development cycle, in part because it is extremely difficult to troubleshoot and is reliant upon the conductor block-io interface which creates a natural performance bottleneck which limits the ability to scale a deployment.

1.9 Wallaby Project Priorities

For the maximum irony!

Of course, we have wants, desires, needs, and hopes. The intention of this document is to convey the priorities for and amongst the community in a published way. Not all of these goals, efforts, and ultimately the features may reach the Wallaby release, nor is it a complete list. It is the list that makes sense to raise visibility of.

This is a list of goals the Ironic team is prioritizing for the Wallaby development cycle, in order of relative size with context of our dependencies and roughly referenced against the anticipated sprints and release cycle for the Wallaby development cycle.

The primary contact(s) listed are responsible for tracking the status of that work and herding cats to help get that work done. They are not the only contributor(s) to this work, and not necessarily doing most of the coding! They are expected to be available on IRC and the ML for questions, and report status on the whiteboard for the weekly IRC sync-up. The number of primary contacts is typically limited to 2-3 individuals to simplify communication. We expect at least one of them to have core privileges to simplify getting changes in.

Note

In the interests of keeping our work fun and enjoyable, while continuing to foster community engagement, this document may have a bit of silliness intertwined. It is all okay, we havent lost all of our sanity, yet.

1.9.1 Goals

Priority	Primary Contacts	Target
Not getting (more) insane	Ironic Developers	Theme
Replacing WSME	stevebaker	Sprint 1
Default to GPT	TheJulia	Sprint 1
Make UEFI happy	dtantsur, TheJulia rpittau	Sprint 1
NVMe Secure Erase	janders, dtantsur, rpittau	Sprint 1
History favors the bold	kaifeng, arne_wiebalck	Sprint 2
Redfish RAID	bdodd, ajya, TheJulia	Sprint 2
Move to oslo.privsep	iurygregory, rpittau	Sprint 2
Configuration molds	rpioso, ajya	Sprint 3
Anaconda deployment	zer0cool, rpittau	Sprint 3
Redfish Interop Profile	arne_wiebalck, rpioso, rpittau	Sprint 3
Snapshots	kaifeng, TheJulia	Future
Security Interface	kaifeng, ljmcgann, rpittau	Future
Boot from URL	Multiple contributors	Future

Schedule Structure

The indicator for this schedule is to help provide those reviewing this document a rough idea of when one may anticipate functionality to merge and be released. Things may merge sooner or later.

Sprint 1

We anticipate the release from the first sprint of the Wallaby cycle to be during the week of December 14th.

Sprint 2

The second sprint starts after the first release and we anticipate the release marking the end of the second sprint to be the week of February 8th, 2021.

Sprint 3

After the second sprint release, the anticipated release date is expected the week of April 5th, 2021.

Theme

General thematic work for general improvement in an area fall under the classification of theme. Largely this is work that may run the course of an entire release cycle or longer, where small incremental improvements or related work takes place.

Future

Items in the future which we as a community do not have a firm idea of *when* this may merge. Being on this list does express that interest exists in the community to push this effort forward during the cycle.

1.9.2 Goals Details

In no particular order

Not getting (more) insane

Ironic is finding increased adoption in use and naturally contributions as needs evolve and new requirements are identified. This is a natural progression. The key that we must keep in mind is that WE can only do so much. We are not super-humans and super human efforts consume the spoons we need to ultimately take over the world.

With this in mind, we must carefully chart our future course. Ultimately we should expect a re-evaluation of our testing matrix, and a focus on what makes the most sense.

Warning

The effectiveness of super-human dresses has not been established in a clinical setting. Special thanks goes to kaifeng for reminding us that we all need to smile. :)

Default to GPT

Ironic supports a number of ways to deploy a physical machine. One of these methods includes use of a partition image. This was defaulted to use a BIOS partition format. But moving forward we need to change to GPT as it is more in line with using UEFI boot when writing partition images for local boot.

Make UEFI happy

UEFI and ultimately Secure Boot being the most community relevant features of UEFI, require doing things in particular patterns which were not well understood nor well documented when the features were ultimately added.

At the same time, there is always a technology adoption lag in data centers and we are beginning to be exposed to various cases and issues where current support is sub-optimal. Ultimately we need to improve this by making Ironic smarter.

At the same time, we likely need to look at making Secure Boot enforceable across drivers. Not all vendors support Secure Boot, but the data center operator interest seems to be substantial at this time.

History favors the bold

To boldly go forth, we must provide more insight into error history of nodes. The concept of adding support to record the important events and surface them in a human parsable way has long been under discussion and been a desired feature. It is time we make it happen.

Node history <https://review.opendev.org/652811> is presently in the review process with strong support from the Project Teams Gathering.

Anaconda deployment

Some operators are invested in Anaconda configurations and using Anaconda kickstart files to facilitate deployments. More information can be found in anaconda deployment specification.

Redfish RAID

Support for using Redfish to configure RAID devices was proposed during the Victoria development cycle but was still in development at the end of the cycle. We hope to see this merged into Ironic during the Wallaby cycle.

NVMe Secure Erase

Ironic needs to do better with more advanced storage devices where secure erase and discard are supported with NVMe devices. The Project Teams Gathering yielded discussion of this, and the possibility of improved support seems likely in the near future.

Snapshots

A major compatibility gap with Novas Compute interaction with VMs that is lacking with Ironic baremetal nodes is support for Snapshots. This is a bit of a complex problem which may require an iterative development process. This is presently under discussion and the community is interested in the functionality. Information about this feature can be found in the snapshot specification document.

Configuration Molds

Configuration molds is the name being given to the conceptual feature of being able to capture the configuration of a machine, and being able to stamp it out across multiple machines. While Ironic has many of these primitives, we do not have the tooling to help enable the easy act of stamping the configuration as a single action. More information can be found in the change 740721.

Move to oslo.privsep

This effort is being carried over from the prior cycle as it became clear the work required would take longer than time existed for us to move the changes forward. More information can be found in the migrate to privsep goal documentation.

Replacing WSME

This work area was started in the Victoria cycle with the initial foundation being put in place, and now it is time to move forward on merging this work.

Most long time contributors are aware of the headaches that WSME has brought the community, along with the fact that many projects have migrated away from it.

In order to move us to something which is supported by a broader community, the consensus from the Train Project Teams Gathering, was to move Ironic towards using Flask. Well start with re-working a single endpoint and hopefully move through the rest of the API in a rapid fashion.

Redfish Interop Profile

Started in the Victoria cycle, the purpose of the interop profile is to declare what is required of a Redfish BMC for our driver to support appropriate management of a baremetal node.

The Redfish Forum has an interop validator utility mechanism to allow BMC vendors to validate their implementation of the Redfish API against the profile that represents compatibility with Ironic.

This work will also enable consumers of hardware to leverage the profile to make sure the hardware they intend to buy works with Ironic or even make this part of their tendering/purchase process.

Security Interface

Recent interest in having an integration with Keylime has brought forth interest in resurrecting the security interface which was proposed some time ago to provide an integration point for Ironic to have the understanding and capability to take the appropriate action in the event a machine has been identified to no longer match the expected profile.

This interface will allow easy adoption of a keylime integration which should allow ironic to halt the return to available inventory of machines which have had unexpected modifications made to firmware.

Boot from URL

This is a long sought after feature, and one more likely to surface as time goes on. Part of the conundrum is the multiple routes possible in what is interpreted as Boot from URL. Luckily Redfish has defined a standard interface to assert the configuration via the BMC.

At a minimum this cycle, we would like to make a step forward in attempting to support this functionality such that we can support it when vendors implement the feature outside of vendor OEM specific mechanisms.

1.10 Victoria Project Priorities

This is a list of goals the Ironic team is prioritizing for the Victoria development cycle, in order of relative size and dependency addressing.

Note that this is not our complete backlog for the cycle, we still hope to review and land non-priority items.

The primary contact(s) listed are responsible for tracking the status of that work and herding cats to help get that work done. They are not the only contributor(s) to this work, and not necessarily doing most of the coding! They are expected to be available on IRC and the ML for questions, and report status on the whiteboard for the weekly IRC sync-up. The number of primary contacts is typically limited to 2-3 individuals to simplify communication. We expect at least one of them to have core privileges to simplify getting changes in.

1.10.1 Goals

Priority	Primary Contacts	Target
Break the Cycle	Ironic Developers	Sprint 1
Move to all native Zuulv3 jobs	iurygregory	Sprint 1
Standalone Basic Authentication	stevebaker	Sprint 1
ISO Boot media pass-through	TheJulia	Sprint 1
Compatibility Matrix	TheJulia	Sprint 1
Replacing WSME	stevebaker	Sprint 2
Ramdisk TLS	dtantsur	Sprint 2
Make CI Manageable	Ironic Developers	Sprint 2
DHCP-less Deployments	iurygregory	Sprint 2
Move to oslo.privsep	iurygregory	Sprint 2
In-band Deploy Steps	dtantsur, mgoddard	Sprint 3
Redfish Interop Profile	arne_wiebalck, rpioso, rpittau	Sprint 3
Bare metal program/SIG	TheJulia, arne_wiebalck	Theme

Schedule Structure

The indicator for this schedule is to help provide those reviewing this document a rough idea of when one may anticipate functionality to merge and be released. Things may merge sooner or later.

Sprint 1

Targeted to be released around the week of June 29th, 2020.

Sprint 2

Starting after the release for Sprint 1.

Anticipated to be released the week of August 10th, 2020. Roughly two months after Sprint 1.

Sprint 3

Starting after the release of Sprint 2, anticipated to run the course of two months. This should be anticipated around the week of September 28th, 2020. It is important to note that the week of the OpenStack deadline for the final release candidate as part of the Victoria cycle. Ironics latest release will be consumed by OpenStack for the Victoria release at this time.

Theme

General thematic work for general improvement in an area fall under the classification of theme. Largely this is work that may run the course of an entire release cycle or longer, where small incremental improvements or related work takes place.

1.10.2 Goals Details

In no particular order

Break the Cycle

Ironic is a project which is consumed by other projects and ultimately makes its way into multiple products through these channels. In many ways, it is a swiss-army knife for operators. At the same time it becomes a semi-hidden implementation detail behind projects like TripleO Metal3, and even in a sense it can be a hidden detail for users of the OpenStack Compute API.

With this use, and ultimately to support their use cases, we need to do better in terms of trying to release early and often.

Sometimes there is no reason TO make a brand new release of Ironic or any of its components unless WE NEED to take action to fix it or we have added a new feature. Largely this is because we rely upon stable APIs and interfaces, at least in most cases.

While we would all love to see all of the features, we should be delivering in a model which conforms to Sem-Ver and not force artificial major version changes to match a cycle boundary and to create a branch point.

Having said this, it is time for us to move to an independent release model.

This has drawbacks, and will require a different approach for planning. And yet not all of Ironics assets should be released in such a way. Some items that fall under our governance as a project have inter-dependencies which ultimately require continuing with the OpenStack release cycle model. Does that make those items any less part of Ironic? Absolutely not! But they are specific integration points, and we will always need to be mindful of the need to maintain those integration points.

Make CI Manageable

CI is a never ending pain point. It is a truth of the existence of a CI system in any software development process. The conundrum we face though, is we have an explosion of drivers, specific use cases we are concerned about, and ultimately we never want to break anyone. The natural end result is we have a huge number of CI jobs.

With a number of different jobs, one may think that it is easy to maintain.

And in a sense, they are right. Except what is lost is our testing model forces `rechecks` to re-run all jobs. It is actually a good thing in that context, yet bad in that any single transient failure can force the re-execution of all tests, again. And Again. And sometimes, yet again.

Ultimately, In order to make CI manageable, we need to improve our concurrency, and greatly reduce our overall job count. We do this by trying to use the same job to test multiple scenarios. Where we cant, we likely have a defect we should explore fixing. And if there is anything we can do to improve job execution time, we ultimately improve time to results for not just ourselves, but everyone.

Move to all native Zuulv3 jobs

The chosen OpenStack community goal for the Victoria cycle is to ensure that all jobs are Zuulv3 native. You can learn about this effort here.

Move to oslo.privsep

One of the proposed, but not selected by the OpenStack TC changes is to adopt the use of Oslo.Privsep. We think this is a good idea and will go ahead and proceed with this effort.

Compatibility Matrix

One of the weaknesses in Ironics documentation is a lack of clarity regarding the functionality of features with-in drivers when comparing drivers side-by-side. Mainly because we want to encourage but not force driver maintainers to take particular improvements to their driver code.

And so, the hope is to fix this weakness in our documentation. Through clarity, the overall user experience should improve, and that is ultimately what we all seek.

In-band Deploy Steps

While a theme that has been part of several of our past cycles, we continue to working towards improving the functionality of Deploy Steps, and in this case the focus is for in-band usage.

Bare metal program/SIG

The most powerful thing the Ironic community can do this cycle is not actually in code, but in documentation. The recently created Bare Metal SIG is working on creation of a white paper as part of the Bare Metal logo program, and needs our help for stand-alone use cases.

Ramdisk TLS

One of the structural weaknesses in the `ironic-conductor` to `ironic-python-agent` security is that by default, it is not encrypted.

In large part because this is a difficult surface to secure. It is ephemeral, temporary, and often short-lived. Mechanisms to sign a certificate are also a bit more difficult to put in place. Does the TLS client check the certificate common name or alias field. Does that even matter for this usage. How do we handle virtual media versus PXE booting with a supplied ramdisk.

These are all questions and concerns that we must answer, with the ultimate goal of ensuring that an agent can automatically offer a TLS encrypted Restful API endpoint for the `ironic-conductor` to connect to.

Replacing WSME

Most long time contributors are aware of the headaches that WSME has brought the community, along with the fact that many projects have migrated away from it.

In order to move us to something which is supported by a broader community, the consensus from the Train Project Teams Gathering, was to move Ironic towards using Flask. Well start with re-working a single endpoint and hopefully move through the rest of the API in a rapid fashion.

Standalone Basic Authentication

For standalone use cases to flourish, we must support another authentication mechanism. The simplest, is rather simple. Just HTTP Basic Authentication.

Maybe we wont just stop there, but noauth is simply not acceptable with edge infrastructure management.

DHCP-less Deployments

Deployment of machines at the edge requires the case where we do not control DHCP. Except there are cases where there might not be any DHCP server, and in such cases, we must supply networking configuration in the virtual media being attached to the physical machine being deployed.

This effort is carried over from the Ussuri development cycle, and with any luck will be merged early in the Victoria development cycle.

ISO Boot media pass-through

A recent idea is to better support the use of the ramdisk use case for operators wishing to trigger machine boot operations via a pre-mastered ISO, much like the existing ramdisk interface, however in this case, they have everything they need.

More information about this can be found in the specification document.

Redfish Interop Profile

The Redfish Forum has an interop profile mechanism to allow feedback in the process to convey what is and what is not supported.

Such a profile can be used by hardware vendors/manufacturers to assess/advertise to which degree Ironic can interact with their hardware. Equally consumers/clients of such hardware can use the profile to make sure the hardware they intend to buy works with Ironic or even make this part of their tendering/purchase process.

1.11 Ussuri Project Priorities

This is a list of goals the Ironic team is prioritizing for Ussuri development cycle, in order of relative size and dependency addressing.

Note that this is not our complete backlog for the cycle, we still hope to review and land non-priority items.

The primary contact(s) listed are responsible for tracking the status of that work and herding cats to help get that work done. They are not the only contributor(s) to this work, and not necessarily doing most of the coding! They are expected to be available on IRC and the ML for questions, and report status on the whiteboard for the weekly IRC sync-up. The number of primary contacts is typically limited to 2-3 individuals to simplify communication. We expect at least one of them to have core privileges to simplify getting changes in.

1.11.1 Goals

Priority	Primary Contacts
Scale / Performance	TheJulia, arne_wiebalck
Bare metal program/SIG	TheJulia, arne_wiebalck
Deploy Steps	dtantsur, mgoddard
Replacing WSME	dtantsur
Node retirement/quarantine	arne_wiebalck, rpittau
Managed boot for inspection	dtantsur
Multitenancy/Machine Ownership	tzumainn
DHCP-less Deployments	etingof

1.11.2 Community Goals

Goal	Primary Contacts
IPv6 support	TheJulia
Drop Python 2.7 Support	iurygregory, rpittau

1.11.3 Experiment for the Future

Ironic is stable, but it doesnt mean we should not experiment and find new ways to solve the same problems. General consensus seems to be that this is an important step forward to allow ourselves to further enable the future.

We give ourself the freedom to Innovate, Experiment, Evolve, and of utmost importance the freedom to listen to our users.

Some of these things may be to just enable Software RAID 5/6, DHCP-less deployment, boot-from-url for IPv4, or even kexec deployment. The idea being that these are focused high-impact and tactical changes and we should encourage such capabilities to merge.

The ultimate goal being to deliver and respond to the needs of our users faster.

1.11.4 Details

Scale / Performance

Ironic is being used all over the world. From small clusters to clusters which may only be able to be described as mind-boggling.

While scale brings a different class of problems, we have consumers and users which can be impacted through even small minor changes. We *must* bring visibility and awareness to this topic as well as ensuring we set expectations and communicate ideal architectures.

- Implement some lightweight stress and performance testing
- Documentation oriented to scaling

It will be important to work with the Bare Metal SIG on this topic.

Deploy Steps

While a theme that was included in the Train cycle, the overall effort for deploy steps is ongoing. The Train cycle provided more interfaces capable of being leveraged via deploy steps. The primary goal is to support in-band execution of deploy steps with the focus being on leveraging Software RAID.

Bare metal program/SIG

The most powerful thing the Ironic community can do this cycle is not actually in code, but in documentation. The recently created Bare Metal SIG is working on creation of a white paper as part of the Bare Metal logo program, and needs our help for stand-alone use cases.

Replacing WSME

Most long time contributors are aware of the headaches that WSME has brought the community, along with the fact that many projects have migrated away from it.

In order to move us to something which is supported by a broader community, the consensus from the Train Project Teams Gathering, was to move ironic towards using Flask. Well start with re-working a single endpoint and hopefully move through the rest of the API in a rapid fashion.

Node retirement/quarantine

Larger operators with Ironic have found themselves approaching a quandry of What is the proper way to retire a machine from ironic?. A nearly identical topic has arisen from the Telecom world seeking to represent the state of the node more accurately with-in ironic as to represent if a machine is in a fault state or questionable state under-going investigation.

With that being said, we did not make progress on this issue in the past cycle due to a fundamental disagreements in perception. The Project Teams Gathering in Shanghai provided a forum for discussion where we realized that these are actually similar but separate issues. One is for a separate logic path in what could be three or six months, and the other is a change in present time.

Managed boot for inspection

In order to support edge architectures and on-demand inspection, we need to enable managing the activation of inspection through ironic.

Multitenancy/Machine Ownership

The reality of hosting environments is that someone owns the hardware. Sometimes this may be the tenant that needs to be on the hardware, and we cant expect them to have administrative access to all hardware.

Thus we need to support a model where a tenant can be granted access a piece of hardware.

DHCP-less Deployments

Deployment of machines at the edge requires the case where we do not control DHCP. Except there are cases where there might not be any DHCP server, and in such cases, we must supply networking configuration in the virtual media being attached to the physical machine being deployed.

IPv6 support

The OpenStack Technical Committee had a goal for the Train cycle for projects to implement IPv6 testing in order to declare IPv6 support. We as a community are aware that our IPv6 support works, however the anticipated changes from a community standpoint were incompatible with the settings required to emulate physical baremetal.

Also, we have encountered some issues with testing IPv6 support where existing default binary builds that are published in distributions lack some of the required support to be enabled.

More information can be found in change 657174.

Drop Python 2.7 Support

Time has come to remove support for Python 2.7 as upstream security support for Python 2.7 is being dropped early in the Ussuri development cycle.

1.12 Train Project Priorities

This is a list of priorities the Ironic team is prioritizing for Train development, in order of relative size and dependency addressing. Note that this is not our complete backlog for the cycle, we still hope to review and land non-priority items.

The primary contact(s) listed are responsible for tracking the status of that work and herding cats to help get that work done. They are not the only contributor(s) to this work, and not necessarily doing most of the coding! They are expected to be available on IRC and the ML for questions, and report status on the whiteboard for the weekly IRC sync-up. The number of primary contacts is typically limited to 2-3 individuals to simplify communication. We expect at least one of them to have core privileges to simplify getting changes in.

1.12.1 Goals

Priority	Primary Contacts
Deploy Steps	mgoddard, rloo
Faster Deployments	TheJulia, dtantsur, stendulker
Bare metal program	TheJulia, janders, hodgepodge
Replacing WSME	mkrai, dtantsur, kaifeng, rpittau
Redfish Virtual Media	etingof, rpittau
Node retirement/quarantine	arne_wiebalck, rpittau
Software RAID	arne_wiebalck, TheJulia

Inter-Project Goals

State callbacks to nova	arne_wiebalck, tssurya
SmartNIC Support	TheJulia, mkrai, moshele

1.12.2 Community Goals

Goal	Primary Contacts
IPv6 support	TheJulia, derekh, dtantsur
Single document generation	kaifeng, rpittau

1.12.3 Details

Deploy Steps

As a general theme of work for the Train Cycle, the Ironic project community wishes to break the monolithic deployment step into multiple deployment steps which will further enable operators to easily create more complex declarative deployments. This work also includes the the ability to trigger steps through the agent, something that is not presently possible today.

Faster Deployments

A repeating theme from the operator community is for ways to speed up the overall deployment time and enable faster time to deployment.

Ironic has completed some work in this area with the Fast Track Deployments, But there are other areas that can and should be explored by the community.

Bare metal program

The most powerful thing the Ironic community can do this cycle is not actually in code, but in documentation. The recently created Bare Metal SIG is working on creation of a white paper as part of the Bare Metal logo program, and needs our help for stand-alone use cases.

Replacing WSME

Most long time contributors are aware of the headaches that WSME has brought the community, along with the fact that many projects have migrated away from it.

In order to move us to something which is supported by a broader community, the consensus from the Train Project Teams Gathering, was to move ironic towards using Flask. Well start with re-working a single endpoint and hopefully move through the rest of the API in a rapid fashion.

Redfish Virtual Media

One of the most powerful features we can offer to operators with distributed and edge ironic nodes is to offer booting the ramdisk via a generic Redfish Virtual Media boot interface. This will enable greater compatibility and once completed in-gate testing of virtual media relate scenarios in CI.

More information can be found in story 1526753.

This work is also a logical step towards the L3 based deployments specification.

Node retirement/quarantine

Larger operators with Ironic have found themselves approaching a quandry of What is the proper way to retire a machine from ironic?. A nearly identical topic has arisen from the Telecom world seeking to represent the state of the node more accurately with-in ironic as to represent if a machine is in a fault state or questionable state under-going investigation.

With that being said, we feel that we need to extend our states and state machine to better fit these overall themes.

Software RAID

Software RAID support has been long desired after by larger operators to help manage COTS server hardware where Hardware RAID controllers are undesirable or prohibitive. Work started during the Stein cycle to support this functionality, and work continues! You can learn more in the Software RAID specification.

State callbacks to nova

One of the headaches and performance issues for larger operators is the nature of power synchronization when nova is in use, as nova performs a large number of API calls to update its database with node power state. At larger scales, this is inefficient and results in the power state nova having on record from being out of state from ironic as the source of truth.

Conversely, nova presently assumes that it is always authoritative in regards to power states. This work will allow ironic to inform nova of the new power state such that nova does not attempt to reset the power state.

While this is largely an effort in the nova project, we need to be aware and attempt to support this work to move forward. The nova-spec document can be found in review as change 636132.

SmartNIC Support

Smartnics complicates ironic as the NIC needs to be programmed with the power in a state such that the configuration on the NIC can be changed.

While the work in Ironic was completed this past cycle ahead of expectations. Work is on-going in Neutron this cycle to merge the functionality to make this available to users.

The story can be found at story 2003346.

IPv6 support

The Technical Committee is presently finalizing a goal for the Train cycle for projects to support and test IPv6-only deployments.

More information can be found in change 657174.

Single document generation

A goal from the Technical Committee is for each project to support the generation of a single PDF document for the whole of the documentation tree.

More information on this community goal can be in governance pdf doc generation goal documentation.

1.13 Stein Project Priorities

This is a list of development priorities the Ironic team is prioritizing for Stein development, in order of relative size and dependency addressing. Note that this is not our complete backlog for the cycle, we still hope to review and land non-priority items.

The primary contact(s) listed is/are responsible for tracking the status of that work and herding cats to help get that work done. They are not the only contributor(s) to this work, and not necessary doing most of the coding! They are expected to be available on IRC and the ML for questions, and report status on the whiteboard for the weekly IRC sync-up. The number of primary contacts is typically limited to 2-3 individuals to simplify communication. We expect at least one of them to have core privileges to simplify getting changes in.

As the time remaining in the Stein cycle is approximately 30 weeks from the Project Teams Gathering, the list of priorities has been split into two major pieces based upon an estimate of relative size. The overall goal is for the *Smaller Goals* items to be focused on with in the first few months of the cycle, while the larger *Epic Goals* may receive some work early on, but will be targeted for later in the cycle.

1.13.1 Smaller Goals

Priority	Primary Contacts
Upgrade Checker	TheJulia, rloo
Python3 First	derek, TheJulia
iPXE/PXE interface split	TheJulia, stendulker
UEFI First	hshiina
HTTPClient booting	TheJulia
Nova conductor_group awareness	jroll, TheJulia
Enhanced Checksum Support	jroll, kaifeng
DHCP-less/L3 virtual media boot	shekar, stendulker

1.13.2 Epic Goals

Goal	Primary Contacts
Deploy Templates	mgoddard, dtantsur, rloo
Graphical Console	mkrai, etingof
Federation Capabilities	TheJulia, dtantsur
Task execution improvements	etingof, TheJulia, mgoddard
No IPA to conductor communication	jroll, rloo
Getting steps	TheJulia, dtantsur
Conductor role splitting	jroll, dtantsur
Neutron Event Processing	vdrok, mgoddard, hjensas

Inter-Project Goals

Deployment state callbacks to nova	TheJulia, jroll
Smartnic Support	TheJulia, mkrai, moshele

1.13.3 Details

Upgrade Checker

This is an OpenStack Community goal for the Stein Cycle. For ironic this will mean a new command called `ironic-status upgrade check`. This command is intended to return an error for things that would be fatal for an upgrade such as new required configuration missing, or schema/data upgrades not yet performed. The story can be found at story 2003657.

Python3 First

This is an OpenStack Community goal for the Stein Cycle. Most of this work has already been completed in ironic. Largely we need to change our tests so we are explicitly testing on Python3. We cant do this for every test at the moment, but we should be able to change most and still ensure the bulk of the code paths are covered by tests labeled with `python2`.

We also desire for third party CI to begin to leverage Python3, with a goal of approximately 50% of third party CI jobs until we stop supporting Python2. The story can be found at story 2003230.

iPXE/PXE interface split

This is an older effort that has been restarted in the interest of supporting multiple architectures (such as AArch64, Power, and x86_64) in the same deployment.

As it turns out, Powers architecture expects the older PXELinux style templates that are written by our PXE boot interface. Additionally, while AArch64 can be booted using iPXE, no pre-built binaries are available.

As such, we need to no longer make this global for the conductor, but specific to the node, and splitting the interfaces apart begins to make much more sense. The original specification can be found ipxe-boot interface. The story can be found at story 1628069.

UEFI First

2020 is an important year for Baremetal Operators, as Legacy boot mode support is anticipated to be removed from newer processors being shipped.

To ensure our success, we need to improve our testing and prepare for the time when UEFI is the only boot mode available for newer hardware. As a result, this will become a multi-cycle focus to enable the default boot mode to be changed to `uefi` in a future cycle. The story can be found at story 2003936.

HTTPClient Booting

While the community is interested in supporting HTTPClient based booting, we currently have a few steps to surpass first. Namely the iPXE/PXE interface split and improved UEFI testing.

The nature of this work is to enable an explicit HTTP booting scenario where the booting node does not leverage PXE. The story can be found at story 2003934.

Nova conductor_group awareness

This work is exclusively in the ironic virt driver in the *openstack/nova* repository. This would enable us to define a `conductor_group` to which the nova-compute process leverages for the view of baremetal nodes it is responsible for. The story can be found at story 2003942.

Enhanced Checksum Support

Ironic presently defaults to use of MD5 checksums for the `image_checksum` which is far from ideal. During the Rocky cycle, Glance has enhanced their support for checksum storage, which means we should enhance ours as well. The story can be found at story 2003938.

DHCP-less/L3 virtual media boot

Some operators and vendors wish to enable ironic to manage deployments where DHCP is not something that is leveraged or utilized in the deployment process. In order to do this, we need to enable some additional capabilities in terms of enabling information to be attached to a deployment ramdisk. The specification can be found at the L3 based deployments specification. The story can be found at story 1749193.

Deploy Templates

In the future, we want to take specific action based upon traits submitted to ironic from Nova describing the instances expected state or behavior.

This will allow us to take actions and influence the deployment steps, and as such is a continuation of the Deploy Steps work from the Rocky cycle. The story can be found at story 1722275.

Graphical Console

We need a way to expose graphical (e.g. VNC) consoles to users from drivers that support it. We reached agreement on the specification in the Rocky cycle and have started to work through the patches to enable this. Our goal being to have a framework and preferably at least one vendor driver to support Graphical console connectivity. The specification can be found vnc graphical console specification. The story can be found at story 1567629.

Federation Capabilities

Edge computing is bringing a variety of cases where support for federation of ironic deployments can be useful and extremely powerful.

In order to better support this emerging use case, we want to try and agree on a viable path forward that meets several different use cases and requirements. The objective for this effort is an agreed upon specification. The story can be found at story 2001821.

Task execution improvements

We realize that our task execution and locking model is problematic, and while it does scale in some ways, it does not scale in other ways. This work will consist of worker execution improvements, an evaluation and possible implementation of different worker thread execution models, and careful improvement of locking. The story can be found at story 2003943.

No IPA to conductor communication

Larger operators need much more strict security in their deployments, where they wish to prevent all outbound network connectivity to the control plane. Presently the design model requires that nodes are able to reach ironics API in order to perform heartbeat and lookup operations.

The concept with this is to optionally enable the conductor to drive the deployment by polling IPA using the already known IP address. That being said, this is realistically going to require *Task execution improvements* to be complete to help ensure that operators are able to have performant deployments. The specification can be found at change 212206. The story can be found at story 1526486.

Getting steps

One of the biggest frustrations that people have with our cleaning model is the lack of visibility into what steps they can execute. This is further compounded with `deploy steps`. We have ideas on this and we need to begin providing the mechanisms to raise that visibility.

This may also involve state machine states to enable the agent to sit in a holding pattern pending operator action.

The goal is ultimately to provide a CLI for the user to be able to understand the available steps that can be utilized. The story can be found at story 1715419.

Neutron Event Processing

Currently ironic has no way to determine when certain asynchronous events actually finish in neutron, and with what result. Nova, on the contrary, uses a special neutron driver, which filters out notifications and posts some of them to a special nova API endpoint. We should do the same. The story can be found at story 1304673.

Conductor role splitting

The conductor presently does all of the work But does it need to?

This is a question we should be asking ourselves as we evolve, if we can optionally break the conductor into many pieces, to enable edge conductors, or edge local boot management. The goal here is to try and obtain a matrix of distinct actions taken, which will hopefully further guide us as time moves on. The story can be found at story 2003940.

Smartnic Support

Smartnics complicates ironic as the NIC needs to be programmed with the power in a state such that the configuration on the NIC can be changed.

While the effort to support this may ultimately result in enhancements to neutron in the form of Super-Agents to apply the configuration, we still need to understand the impact to our workflows and ensure that sufficient security is still present. The primary objective is to have a joint specification written in advance of the Berlin summit to reach consensus with the Neutron team as to the mechanics, information passing, and setting storage. The story can be found at story 2003346.

Deployment state callbacks to nova

One of the issues in ironics nova virt driver is that no concept of callbacks exist. Due to this, the virt driver polls the ironic API endpoint repeatedly, which increases overall system load. In an ideal world, ironic would utilize a mechanism to indicate deployment state similar to how neutron informs nova that networking has been configured. The story can be found at story 2003939.

1.14 Rocky Project Priorities

This is a list of development priorities the Ironic team is prioritizing for Rocky development, in order of priority. Note that this is not our complete backlog for the cycle, we still hope to review and land non-priority items.

The primary contact(s) listed is/are responsible for tracking the status of that work and herding cats to help get that work done. They are not the only contributor(s) to this work, and not necessary doing most of the coding! They are expected to be available on IRC and the ML for questions, and report status on the whiteboard for the weekly IRC sync-up. The number of primary contacts is typically limited to 2-3 individuals to simplify communication. We expect at least one of them to have core privileges to simplify getting changes in.

1.14.1 Priorities

Priority	Primary Contacts
Deploy Steps	rloo, mgoddard
BIOS config framework	zshi, yolanda, moddard, hshiina
Conductor Location Awareness	jroll
Reference architecture guide	dtantsur, jroll
Graphical console	mkrai, anup-d-navare, TheJulia
Neutron Event Processing	vdrok

1.14.2 Goals

Goal	Primary Contacts
Updating nova virt to use REST API	TheJulia
Storyboard migration	TheJulia, dtantsur
Management interface refactoring	etingof, dtantsur
Getting clean steps	rloo, TheJulia
Project vision	jroll, TheJulia
SIGHUP Support	rloo

1.14.3 Stretch Goals

Note

Upon completion of *Storyboard migration*, the stretch goals documented here will be migrated to and tracked in Storyboard.

Stretch Goals	Primary Contacts
Classic driver removal	dtantsur
Redfish OOB inspection	etingof, deray, stendulker
Zuul v3 playbook refactoring	sambetts, pas-ha

1.14.4 Details

Deploy Steps

We created cleaning with the ability to compose an ordered list of actions to be taken. However we left deployment as a static set of actions.

In order to allow templates to apply using chosen traits, we want to have the same functionality and framework that we obtained with cleaning to apply to the deployment of a node.

This will start with the goal of splitting apart the action of writing the image during the deployment from the act of writing the configuration drive into two distinct steps. From there, we will further iterate.

This may ultimately facilitate de-duplication of deployment logic which was an uncompleted goal from the Queens development cycle.

BIOS config framework

Some drivers support setting BIOS (UEFI, etc) configuration out-of-band. We would like to introduce a framework (HTTP and driver API) for drivers to expose this feature to users.

Conductor location awareness

Often operators have made changes to driver names to facilitate mapping of conductors to individual nodes, such that conductors are local to nodes, and a conductor in Los Angeles is not trying to control machines in Europe.

This allows ironic to remain a single pane of glass, and provides increased flexibility in deployments of ironic. For now, we will focus on hard affinity with an upgrade path.

Reference architecture guide

To help new deployers make the right choices, we need a document describing a reference architecture for an ironic deployment, especially around multi-tenant networking and co-existing with VMs.

Graphical console

We need a way to expose graphical (e.g. VNC) consoles to users from drivers that support it. Specifications and patches are in various states, and need to be picked up again. We are hoping to have the initial framework to support graphical console usage in this cycle.

Neutron event processing

Currently ironic has no way to determine when certain asynchronous events actually finish in neutron, and with what result. Nova, on the contrary, uses a special neutron driver, which filters out notifications and posts some of them to a special nova API endpoint. We should do the same.

Updating nova virt to use REST API

Cycle after cycle we encounter issues with the upgrade sequence and the API version pin that we increment in the Nova virt driver that supports ironic.

It is time to put this to an end and better properly support versioned interactions with our API. Upon discussing with the Nova community, we reached consensus that it is time to transition our API calls to direct REST calls as opposed to calls through the python-ironicclient library.

Storyboard migration

Ironic has planned work in many different ways over the years, and we have learned that there are positives to each approach.

As our team has also become smaller, we must also enable better focus by integrating distinct tools, systems, and processes together.

With Storyboard, we will gain the advantage of planning and tracking work in a single system.

Management interface refactoring

As time has gone on, we have found the need to have a single place to control the boot mode for nodes. This effort is refactoring the management interface so we move distinct boot mode related actions into a single interface.

Getting clean steps

One of the biggest frustrations that people have with our cleaning model is the lack of visibility into what they can do. We have ideas on this and we need to begin providing the mechanisms to raise that visibility.

Project vision

We all have different ideas of where we would like to see ironic in two, five, and ten years. Discussing this as a group helped us scope and frame our discussions so we were on the same page.

We should write down our collective vision of the future, and see where it takes us.

SIGHUP support

SIGHUP is the signaling mechanism to indicate that a program should attempt to reload configuration and possibly restart itself. Supporting SIGHUP is an OpenStack project wide goal, and it should be easy for us. Lets do it!

Classic driver removal

We have deprecated the classic drivers, and soon is approaching the time to remove these drivers now that we have provided a means to migrate users to hardware types. Deprecation took place on Feb 1, 2018, and thus this code can be removed after May 1, 2018.

Redfish OOB inspection

Redfish is one of our in-tree reference hardware types, however we have no support for out-of-band inspection. In terms of providing feature parity, we should move forward with this, as more vendors are moving to Redfish.

Zuul v3 playbook refactoring

One of the powerful features with Zuul v3 is that we execute ansible playbooks as opposed to traditional shell scripting. The migration left quite a bit of legacy shell scripts in the testing process.

Efforts are underway to remove the bulk of this launch scripting from our normal devstack jobs. We should expect our grenade jobs to remain untouched.

1.15 Queens Project Priorities

This is a list of development priorities the Ironic team is prioritizing for Queens development, in order of priority. Note that this is not our complete backlog for the cycle, we still hope to review and land non-priority items.

The primary contact(s) listed is/are responsible for tracking the status of that work and herding cats to help get that work done. They are not the only contributor(s) to this work, and not necessary doing most of the coding! They are expected to be available on IRC and the ML for questions, and report status on the whiteboard for the weekly IRC sync-up. The number of primary contacts is limited to 2-3 maximum to simplify communication. We expect at least one of them to have core privileges to simplify getting changes in.

1.15.1 Essential Priorities

Priority	Primary Contacts
Ironic client version negotiation	TheJulia, dtantsur
External project auth rework	pas-ha, TheJulia
Old ironic CLI deprecation	rloo
Classic drivers deprecation	dtantsur
Reference architecture guide	dtantsur, sambetts
Traits support planning	johnthetubaguy, TheJulia, dtantsur, mgoddard

1.15.2 High Priorities

Priority	Primary Contacts
Neutron event processing	vdrok, vsaienk0, sambetts
Routed networks support	sambetts, vsaienk0, bfournie
Rescue mode	rloo, stendulker, aparnav
Clean up deploy interfaces	vdrok
Zuul v3 jobs in-tree	sambetts, derekh, jlvillal
Graphical console	pas-ha, vdrok, rpioso
BIOS config framework	dtantsur, yolanda, rpioso
Ansible deploy interface	pas-ha, yuriyz

1.15.3 Details

Ironic client version negotiation

Currently, we only support using a single API version for the whole life time of an ironic client instance. We need to support using several versions in the same client. We also need a way for the client to negotiate a maximum mutually supported version with a server. Then we need to switch to using this negotiated version in the OSC CLI by default. Finally, we will use the version negotiation in the ironic virt driver.

External project auth rework

Work is under way to change how ironic authenticates with other OpenStack services, so that it uses keystoneauth adapters. This will make it uniform, consistent, straightforward, and compatible with the latest OpenStack best practices.

Old ironic CLI deprecation

We would like to deprecate the `ironic` CLI tool in favour of `openstack baremetal` commands based on OpenStackClient.

Classic drivers deprecation

We would like to deprecate the ability to load classic drivers, as well as classic drivers themselves, in favour of hardware types. We are providing a migration guide for the switch, and we are also considering writing an automated migration tool, though it may be tricky. See *Future of classic drivers* for details.

Reference architecture guide

To help new deployers make the right choices, we need a document describing a reference architecture for an ironic deployment, especially around multi-tenant networking and co-existing with VMs.

Neutron event processing

Currently ironic has no way to determine when certain asynchronous events actually finish in neutron, and with what result. Nova, on the contrary, uses a special neutron driver, which filters out notifications and posts some of them to a special nova API endpoint. We should do the same.

Routed networks support

Ironic should become aware of L2 segments available to connected networks as well as which L2 networks are actually available to nodes to correctly pick subnet (IP address) when doing provisioning/cleaning.

Rescue mode

This is necessary for users that lose regular access to their machine (e.g. lost passwords). The *rescue mode spec* was merged in Newton, the code is partially done, lets put some effort into finishing it in Queens.

Clean up deploy interfaces

There is a lot of duplication between the `iscsi` and `direct` deploy interface implementations. We need to clean them up to simplify future maintenance.

Zuul v3 jobs in-tree

With the switch to Zuul v3 we can now have our job definitions in our source tree. Let us switch all jobs during this cycle.

Graphical console

We need a way to expose graphical (e.g. VNC) consoles to users from drivers that support it.

BIOS config framework

Some drivers support setting BIOS (UEFI, etc) configuration out-of-band. We would like to introduce a framework (HTTP and driver API) for drivers to expose this feature to users.

Ansible deploy interface

A deploy interface using ansible was developed out-of-tree and is currently a part of ironic-staging-drivers. We need to import it into ironic to simplify advanced use cases, requiring extensive customizations. The *spec* was approved, now we need to clean up the code and move it in-tree.

Traits support planning

Nova is switching from *capabilities* to *traits* in the coming cycles. We should make sure we are ready for the switch. The minimum goal for Queens is to have a specification approved, outlining our plan on traits support.

1.16 Pike Project Priorities

This is a list of development priorities the Ironic team is prioritizing for Pike development, in order of priority. The primary contact(s) listed is/are responsible for tracking the status of that work and herding cats to help get that work done. They are not the only contributor(s) to this work! The number of primary contacts is limited to 2 maximum to simplify communication. We expect at least one of them to have core privileges to simplify getting changes in.

1.16.1 Essential Priorities

Priority	Primary Contacts
Standalone CI tests	vsaienk0
Generic boot-from-volume	TheJulia, dtantsur
Rolling upgrades	rloo, jlvillal
Reference architecture guide	jroll, dtantsur
Python 3.5 compatibility	Nisha
Deploying with Apache and WSGI in CI	vsaienk0
Driver composition	jroll, dtantsur
Feature parity between two CLIs	rloo, dtantsur
OSC default API version change	dtantsur
Finish node tags	zhenguo, dtantsur

1.16.2 High Priorities

Priority	Primary Contacts
Rescue mode	stendulker, aparnav
Post-deploy VIF attach/detach	sambetts, vsaienk0
Physical network awareness	sambetts, vsaienk0
Routed networks support	sambetts, vsaienk0
Neutron event processing	vdrok, vsaienk0
IPA REST API versioning	sambetts

1.16.3 Optional Priorities

Priority	Primary Contacts
Split away the tempest plugin	soliosg, jlvillal
Deploy steps	yolanda, rloo
Redfish driver	lucasagomes, jroll
Supported power states API	dtantsur
Available clean steps API	rloo
E-Tags in API	galyna, vdrok

1.16.4 Details

Standalone CI tests

We are working on a set of tests that can be run without nova present. We expect this work to improve CI time, coverage and help test certain features (e.g. adoption) that are not quite compatible with nova.

Generic boot-from-volume

This work allows generic hardware to boot from remote storage, allowing diskless nodes to be managed by ironic. This also lays down the framework for hardware-specific implementations to be built.

Rolling upgrades

Many OpenStack projects are beginning to support rolling upgrades - we should too. Lets do our part to make downtimes a thing of the past. This involves code changes, new multi-node grenade CI jobs, and reviewer/developer documentation. The target now is Ocata -> Pike rolling upgrades.

Reference architecture guide

To help new deployers make the right choices, we need a document describing a reference architecture for an ironic deployment, especially around multi-tenant networking and co-existing with VMs.

Driver composition

Weve done most of the coding. Lets concentrate on stabilizing the feature, writing new hardware types, polishing documentation and helping vendors with onboarding.

Feature parity between two CLIs

Let us make sure that all features implemented in the old `ironic` CLI are available in the new OSC-based `openstack baremetal` CLI.

OSC default API version change

Currently the default API version OSC sends to ironic is an old Kilo-era one. We need to figure out the path towards making the latest version the default.

Rescue mode

This is necessary for users that lose regular access to their machine (e.g. lost passwords). The spec was merged in Newton, the code is partially done, lets put some effort into making progress here in Pike.

Post-deploy VIF attach/detach

We already support attaching and detaching VIFs to nodes as part of the deployment process. Now we need to support the same for active nodes.

Physical network awareness

We need to make sure instances are not scheduled on nodes that cannot physically reach the networks theyre connected to. This may require data model changes around ports. This is required for *Routed networks support*.

Routed networks support

Ironic should become aware of L2 segments available to connected networks as well as which L2 networks are actually available to nodes to correctly pick subnet (IP address) when doing provisioning/cleaning.

Neutron event processing

Currently ironic has no way to determine when certain asynchronous events actually finish in neutron, and with what result. Nova, on the contrary, uses a special neutron driver, which filters out notifications and posts some of them to a special nova API endpoint. We should do the same.

IPA REST API versioning

IPA API is currently not versioned, which causes problems when ironic starts relying on new features. Versioning similar to ironic API is expected to fix it and simplify upgrades.

Split away the tempest plugin

Currently we rely on certain hacks to make our CI use master version of the tempest plugin on all branches. The QA team suggests moving our tempest plugin to a separate branch instead, lets do it. We should also merge ironic and ironic-inspector plugins for simpler maintenance and consumption.

Deploy steps

This is an effort to split parts of our monolithic deployment process into steps, similar to cleaning. That will give driver authors a bit more freedom in customizing the deploy process, and simplify potential additions to it (like RAID, for example).

Supported power states API

The soft power/NMI spec proposes exposing available power states in the API. We didnt implement this part in Ocata, let us finish it now.

Available clean steps API

We need to expose available clean steps in the API, so that users know which actions they can run during manual cleaning. This is a part of the manual cleaning spec which was never implemented, despite the spec being marked as done.

E-Tags in API

We should add E-Tag support to our API to avoid race conditions during concurrent updates.

1.17 Ocata Project Priorities

This is a list of development priorities the Ironic team is prioritizing for Ocata development, in order of priority. The primary contact listed is responsible for tracking the status of that work and herding cats to help get that work done.

Priority	Primary Contacts
Portgroups support	sambetts, vdrok
CI refactoring	dtantsur, lucasagomes
Rolling upgrades	rloo, jlvillal
Security groups	jroll
Interface attach/detach API	sambetts
Generic boot-from-volume	TheJulia
Driver composition	dtantsur
Rescue mode	JayF
Smaller things	jroll/all

1.17.1 Portgroups support

This has been in progress for a couple cycles now and is almost complete. It allows for using bonding to support a single connection over multiple NICs. Lets get it done in Ocata.

1.17.2 Rolling upgrades

Many OpenStack projects are beginning to support rolling upgrades - we should too. Lets do our part to make downtimes a thing of the past. This involves code changes, new multi-node CI jobs, and reviewer/developer documentation.

1.17.3 Security groups

This covers security groups on the provisioning network(s). It barely missed the Newton release and is essentially done, lets finish it in Ocata.

1.17.4 Interface attach/detach API

This makes our API to attach a Neutron port to an Ironic port object a first-class API, rather than pushing data into a JSON blob. It also refactors how Nova attaches these ports together. This is a pre-requisite for VLAN-aware instances, so lets get it done and unblock that for Pike.

1.17.5 Generic boot-from-volume

This work allows generic hardware to boot from cinder volumes or NFS, allowing diskless nodes to be managed by ironic. This also lays down the framework for hardware-specific implementations to be built.

1.17.6 Driver composition

This work refactors the way that drivers are composed internally, as well as allowing operators to mix and match drivers for each interface rather than guessing at which driver is which combination. This allows us to stop exploding our driver matrix with every interface addition.

1.17.7 Rescue mode

This is necessary for users that lose regular access to their machine (e.g. lost passwords). The spec was merged in Newton, the code is partially done, lets put some effort into making progress here in Ocata.

1.17.8 CI refactoring

We have too many jobs, and we have a plan to consolidate those jobs. We need to do that ASAP, as infra isnt interested in adding more jobs for us until then.

1.17.9 Smaller things

Theres a number of smaller changes wed like to accomplish in Ocata, or larger things that are nearly code-complete. These include:

- etags in the REST API
- spec for ironic-python-agents REST API versioning
- spec for deploy steps
- spec for specific fault support
- adding more notifications

- soft power off and NMI support
- node tags

1.18 Newton Project Priorities

This is a list of development priorities the Ironic team is prioritizing for Newton development, in order of priority. The primary contact listed is responsible for tracking the status of that work and herding cats to help get that work done.

Priority	Primary Contacts
Upgrade testing	jlvillal, mgould
Network isolation	jroll, TheJulia, devananda
Gate improvements	jlvillal, lucasagomes, dtantsur
Node search API	jroll, lintan, rloo
Node claims API	jroll, lintan
Multiple compute hosts	jroll
Generic boot-from-volume	TheJulia, dtantsur, lucasagomes
Driver composition	dtantsur

1.18.1 Upgrade testing

We claim to support upgrading Ironic from release to release (and cycle to cycle), however we dont have testing to prove that. This item is specifically to get cold upgrade testing working, which is necessary for some of the heavier changes, like network isolation and multi-compute. As a note, weve agreed that this is important enough to block other work.

1.18.2 Network isolation

This feature was designed in Liberty, and much of the code was written. The code was not ready in time to land in Mitaka. We need to complete this work in Newton, along with the Nova side of the work. This is one of our biggest feature asks from users. To be clear, the priority is to land the parts that were already designed in Liberty/Mitaka, not the future work like vlan-aware-instances.

1.18.3 Gate improvements

There exist other gaps in our gate testing, as well as some refactoring we wish to do. This includes:

- Switching to tinyipa instead of the CoreOS image.
- Switching to virtualbmc instead of the SSH driver.
- Testing local boot
- Testing agent driver with partition images
- Getting grenade-partial running to test live upgrades (even if it isnt stable yet)

1.18.4 Node search API

This lays the groundwork for the work being done in Nova to allow the Ironic driver to utilize multiple compute hosts. The search API also helps users query nodes much more intelligently.

1.18.5 Node claims API

This lays the groundwork for the work being done in Nova to allow the Ironic driver to utilize multiple compute hosts. The claims endpoint will help clients other than Nova schedule to nodes more easily.

1.18.6 Multiple compute hosts

This is an effort to allow the Ironic virt driver in Nova scale across many compute hosts. Currently only one compute host is supported. This shrinks the failure domain of the nova-compute service in an Ironic deployment, and also helps schedule Ironic resources more efficiently. Note that this work is in the Nova codebase, but is an Ironic effort. This will likely depend on work happening in Nova, specifically the generic resource pools work in the scheduler.

1.18.7 Generic boot-from-volume

This work allows generic hardware to boot from NFS or cinder volumes, allowing diskless nodes to be managed by ironic. This also lays down the framework for hardware-specific implementations to be built.

1.18.8 Driver composition

This work refactors the way that drivers are composed internally, as well as allowing operators to mix and match drivers for each interface rather than guessing at which driver is which combination. This allows us to stop exploding our driver matrix with every interface addition.

1.19 Mitaka Project Priorities

This is a list of development priorities the Ironic team is prioritizing for Mitaka development, in no particular order.

You may notice this list seems long for a single cycle. Note that much of this work is in flight already and just needs to be completed during the Mitaka cycle; they are not starting from scratch.

Priority	Primary Contacts
ironic-lib refactor	rloo
Manual cleaning	rloo
RAID	rameshg87, lucasagomes
Network isolation	jroll
Live upgrades	lucasagomes, lintan
Boot interface refactor	jroll
Parallel tasks with futurist	dtantsur
Node filter API and claims endpoint	jroll, devananda
Multiple compute hosts	jroll, devananda
Improving testing	jlvillal, krtaylor, lekha
Improving docs	jroll, liliars

May 2016. These were completed in the Mitaka cycle:

- *ironic-lib refactor*
- *Manual cleaning*
- *RAID*
- *Boot interface refactor*

- *Parallel tasks with futurist*

1.19.1 ironic-lib refactor

Work to refactor much of Ironics disk partitioning code into a library (ironic-lib) began in Liberty, getting the library set up and released. The team would like to finish that work in Mitaka by removing the code in Ironic and replacing it with library calls. This enables advanced partitioning in ironic-python-agent via this library, which will be needed for supporting partitioned disk images.

1.19.2 Manual cleaning

Work began on a feature called zapping in Liberty. This came late in the cycle, and the implementation identified some issues with the architecture and name. This is being re-spun as manual cleaning in Mitaka. Its needed to complete the RAID work and for operators to be able to trigger other cleaning tasks manually.

1.19.3 RAID

Much of the groundwork was laid down for RAID support in Liberty. To complete this work in Mitaka, we need to integrate it with (manual) cleaning and write ample documentation for operators.

1.19.4 Network isolation

This feature was designed in Liberty, and much of the code was written. The code was not ready in time to land in Liberty. We need to complete this work in Mitaka, along with the Nova side of the work. This is one of our biggest feature asks from users.

1.19.5 Live upgrades

This is necessary, especially with frequent releases, to allow our operators to upgrade without downtime. There isnt much code work left to do here; mostly docs, increasing awareness, and building a culture of coding and reviewing for live upgrades.

1.19.6 Boot interface refactor

This work was completed for most drivers in Liberty. Two drivers (iLO and iRMC) remain to be completed. The code is complete for this and just needs review.

1.19.7 Parallel tasks with futurist

This will help scale our conductor service, as we have a handful of periodic tasks that currently run serially. Additionally, drivers may register periodic tasks that compound this problem.

1.19.8 Node filter API and claims endpoint

This lays the groundwork for the work being done in Nova to allow the Ironic driver to utilize multiple compute hosts. The filter API also helps users query nodes much more intelligently, and the claims endpoint will help clients other than Nova schedule to nodes more easily.

1.19.9 Multiple compute hosts

This is an effort to allow the Ironic virt driver in Nova scale across many compute hosts. Currently only one compute host is supported. This shrinks the failure domain of the nova-compute service in an Ironic deployment, and also helps schedule Ironic resources more efficiently. Note that this work is in the Nova codebase, but is an Ironic effort.

1.19.10 Improving testing

There are a number of gaps in our testing that we need to close. This includes full tempest, microversion testing, grenade jobs, functional testing, third party CI, and more. These will help us to keep our releases more stable.

1.19.11 Improving docs

We currently dont have any presence in the official OpenStack documentation, and somewhat related, there is currently no way to update documentation for stable branches. We also need to create a developers guide of sorts, to help developers follow our processes more easily and submit better code, as well as help reviewers review code in a consistent manner.

CHAPTER

TWO

SPECIFICATIONS

Specifications for the Ironic project are available here. Specifications begin life in the approved tree. They stay there (possibly beyond the development cycle in which they had been approved), and once implemented, are moved to the implemented tree for that development cycle. Additionally, a backlog of ideas is maintained to indicate the agreed-upon goals for the project which have no specific work being done on them at this time.

CHAPTER

THREE

APPROVED SPECIFICATIONS

These specifications have been approved but have not been completely implemented:

3.1 Add a field to accept the default verify_ca path

https://bugs.launchpad.net/ironic/+bug/2040236

This spec adds an option for setting [driver]verify_ca for each driver. This value would then be used by Ironic for certificate validation instead of the default system CA bundle for that driver. Each driver may use a different CA certificate for this purpose if desired.

3.1.1 Problem description

Currently, Ironic utilizes a system-wide CA certificate bundle to validate HTTPS connections. However, there are scenarios where this default behavior may not align with the specific requirements or policies of an organization. Particularly, when a user specifies that certificate validation is required, administrators may prefer not to rely on the systems default certificate path for authentication with BMCs. Instead, they might need to utilize custom CA certificates, which are tailored to their organizational security policies or specific use cases.

At present, if the operator desires to configure a custom CA certificate for communications between Ironic and the BMCs of managed bare-metal servers, they have the ability to configure the <driver>_verify_ca option on the node or nodes in question. This option can be configured to be either true, false, or set to a specific certificate path. When <driver>_verify-ca is set to true, Ironic defaults to using the systems CA bundle for certificate validation. Furthermore, in cases where the operator desires to specify a CA certificate outside system CA bundle, this approach has a serious shortcoming: it requires the operator to know the exact location of the pre-existing CA certificate in the filesystem. While this configuration option is set on per-node level through Ironic APIs, there is no way for the operator to either upload the CA certificate through those APIs or to determine the filesystem path of such certificate. This is not a desirable situation. This option is also unsuitable for the use cases where the operator is only able to interact with Ironic through writing configuration file and making API calls (which is the case in metal3).

To overcome this limitation, we propose a new configuration option scoped on a driver level (as opposed to the node level): [driver]verify_ca. This way the required CA certificates can be deployed to known locations at the deploy time and Ironic configuration for each driver can be written referencing those paths.

3.1.2 Proposed change

1. Adding a new option verify_ca for each driver

- Adding a verify_ca option to each drivers conf to accept the specified value. Making this option configurable on a per-driver level is consistent with existing configuration options (e.g. kernel_append_params). It is also very clear what purpose this specific CA certificate serves.

2. Retrieving the path before node verification

- Before performing the node verification, retrieve the certificate path, and pass it to verify_ca for validation. This implementation vary based on different vendors.

Alternatives

- Administrators may need to log in to the system where ironic is located and manually add the desired certificate. Note: this requires modifications of the trusted CA certificate collection on the machine running Ironic, which may or may not be desired; there may be use cases where an operator wishes to trust a certain CA for connections to BMC but not other encrypted communications involving the server running Ironic.
- Instead of [driver]verify_ca, a global configuration option similar to [conductor]verify_ca could be considered, however this could lead to confusion about the impact of setting it (does it apply to all conductor connections? BMC connections? else?) and would be inconsistent with other, similar options (e.g. kernel_append_params).
- Using the existing <driver>_verify_ca setting is another alternative, however it requires prior existence of the desired CA certificate in the filesystem where Ironic is running as well as operators knowledge of the exact path, which should not be relied upon and as such is not desired.

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

None

Security impact

The introduction of the verify_ca configuration option in Ironics driver could have several security implications:

1. Handling of Sensitive Data:

- This change involves the path to CA certificates, which are critical for secure communications. While the option itself doesnt handle sensitive data like tokens or keys, it directs Ironic to the location of sensitive cryptographic material. Proper management and permissions of this path are essential to prevent unauthorized access.

2. Accessibility of Managed Hardware:

- By allowing a custom CA path, this change could affect the security of the hardware managed by Ironic. If an incorrect or untrusted CA path is specified, it could potentially compromise the integrity of the SSL/TLS connections with the BMCs, impacting the secure management of the hardware.

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

- At deploy time, one or more custom CA certificates may be installed on the machine running Ironic under a known path.
- Ironic configuration created at the deploy time will assign the custom CA certificate to the driver(s) that are expected to be using it.
- Each driver can use a different CA certificate, or the same CA certificate may be used by multiple drivers if desired.
- The default None value for verify_ca ensures backward compatibility, using the systems default CA bundle unless overridden. This approach maintains operational stability for existing deployments while offering flexibility for custom configurations.
- Ironic will be enhanced to dynamically retrieve the value of verify_ca for each hardware driver and pass it to the verify_ca function. This mechanism ensures that SSL/TLS communications with BMCs across different hardware types can leverage the specified custom CA certificates.
- The feature requires explicit enablement by deployers. This change will not automatically activate.

Developer impact

- Developers will need to ensure that their drivers correctly interpret and utilize the specified CA path.

3.1.3 Implementation

Assignee(s)

Primary assignee:
Zhou Hao <zhouhao@fujitsu.com>

Other contributors:
Zou Yu <zouy.fnst@fujitsu.com> Feng GuangWen <fenggw-fnst@fujitsu.com>

Work Items

- Implement option verify_ca for each vendor.
- Update documentation.

3.1.4 Dependencies

None

3.1.5 Testing

- Test the driver to verify that the set path is used when performing certificate validation.

3.1.6 Upgrades and Backwards Compatibility

None

3.1.7 Documentation Impact

- The documentation should be updated for each hardware vendor as features are implemented.

3.1.8 References

None

3.2 Attestation Interface and Keylime Integration

https://storyboard.openstack.org/#!/story/2002713

In order to help verify that baremetal nodes are in a trustworthy state, we are in need of an interface that allows us to take certain actions or verification steps while proceeding along the state machine.

Some of these steps may involve calling an external attestation server, or executing a special step during cleaning in order to ensure that a node is owned by the attestation server.

At a high level, we need an interface of hooks. And there is no better way than to provide a facility to execute external tooling.

3.2.1 Problem description

Terms Glossary

In trying to bring together two unrelated services, a bit of namespace pollution was inevitable. So for those unfamiliar with Keylime terminology and to avoid confusion with Openstack vocabulary, we will define all the terms needed for this spec here.

Trusted Platform Module (TPM) - a microcontroller within a machine which can create and store hashes securely. All nodes looking to use Keylime for attestation will need to have TPM 2.0.

Integrity Measurement Architecture (IMA) - A security subsystem in Linux which gathers hashes of files, file metadata, and process metadata as a measurement of the system. Stores the measurement in the machines TPM. In the context of Ironic and Keylime integration, we will need to run IMA on the node we are attesting.

allowlist - A hash representing the golden state of the node. In the context of Keylime, an allowlist is compared with an IMA measurement to see if the node has been tampered with in an unauthorized way.

Keylime verifier - A component of the Keylime suite which is responsible for comparing the allowlist to the measurement gathered from the node we are attesting. The verifier will run on a machine external to Ironic and the node Ironic is controlling and looking to attest.

Keylime registrar - A component of the Keylime suite which Ironic will need to talk to in order to initiate the attestation workflow for a node. The registrar also runs on a machine external to Ironic and the node. The verifier and registrar may run on the same machine, but it is not necessary and the decision is left to the operator.

Keylime agent - A component of the Keylime suite which runs on the node we are attesting. The agent will command IMA to collect measurements and send the measurements to the verifier.

Keylime tenant - An API which the Ironic conductor will need to use to communicate with the registrar and verifier. Not to be confused with Openstack tenants.

Introduction

Presently, we rely upon a certain level of trust for users that leverage baremetal resources. While we do perform cleaning between deployments, a malicious attacker could potentially modify firmware of attached devices in ways that may or may not be readily detectable.

The solution that has been proposed for this is the use of a measured launch environments with engagement of Trusted Platform Management (TPM) modules to help ensure that the running system profile is exactly as desired or approved, by the attestation service.

But from a security standpoint, security is not always about code. Sometimes security is adherence to process. To leverage TPMs for attestation, we propose Keylime, an open source remote boot attestation and runtime integrity measurement system.

The first step requires a new interface type attestation_interface to be added as a subclass of BaseDriver. This would then come with a attestation_interface implementation which would use Keylime to learn about the security state of a node and manage configurations. All calls to the attestation interface would happen along existing clean and deployment workflows and simply fail transition if a node is deemed to be compromised.

The second step is a set of enhancements for the ramdisk to support TPM 2.0, and installation of the Keylime agent. From there the Keylime agent would communicate with the registrar and verifier. The manager would trigger attestations at certain points along the nodes workflow ex) during the boot process. Note that in order to perform attestation, the verifier must be within the same network as the node.

3.2.2 Proposed change

Attestation Interface

The addition of a `attestation_interface` field in the `nodes` table, which maps to a *task.node.driver.attestation* interface, along with the other standard configuration parameters and defaults behavior that exists with the driver composition model.

Accordingly the `attestation_interface` would be returned on the node object when retrieved via the REST API, and will be able to be set as another interface.

The attestation_interface will provide a means of configuring and orchestrating a nodes connection with a verifier machine.

The Ironic controller will work under the assumption that the network used to communicate with the attestation service is secure and that the attestation entity is also always trustworthy. Trying to concern ourselves with issues like replay attacks or spoofed messages is beyond the scope of IMA attestation workflows.

To accommodate operator workflows wherein an operator may not have access to the attestation service, we cannot allow the attestation service perform any orchestration. This requires all communication to an attestation service to involve the Ironic controller polling an API for a status or instructing the attestation service or node to take action, as opposed to receiving information from the attestation service or node itself. For example, Keylime offers revocation frameworks for taking action immediately upon a node being compromised. However, from Ironics perspective, allowing another service to do any orchestration could put Ironic in a state where it does not know what is happening on the node.

Presently, we are mainly concerned with monitoring deployment and cleaning of a node. The intended workflow will be to use the interface during these steps to ensure the firmware of a node has not been modified.

Keylime Interface

The Keylime interface will inherit from the AttestationInterface class. The purpose of the interface is to allow the controller to gather relevant information about the security state of the node and take action based on the results. Doing so will require methods which will make calls to the Keylime verifier through the available REST API as well as calls to the IPA to pass necessary configuration parameters. Keylime is anticipated to be supported by generic hardware types.

Keylime Configuration

The Keylime verifier and Keylime registrar are two components of the Keylime suite which must be stood up by an operator. The verifier and registrar will need TLS connections over https in order to communicate. The Keylime tenant CLI is installed on ironic controller. The operator will be responsible for securing any network the registrar and verifier are setup in.

Detailed communication requirements are list as following:

> Keylime tenant -> Keylime verifier: mutual TLS connection
>
> Keylime verifier/tenant -> node: unencrypted connection
>
> Keylime verifier/node/tenant -> registrar: mutual TLS connection for post/put requests; unencrypted connection for get/delete requests

Every Keylime agent must have a uuid associated with it in order to register itself with the registrar. It generates its uuid using the Keylime config file. The uuid defaults to a random id.

Allowlist and Excludelist

Allowlists and Excludelists will be generated beforehand and hosted on a remote server or in the conductors filesystem. A filepath for the conductors filesystem or url to a remote server to locate such files will be supplied to Ironic before provisioning. Allowlists may also be signed with a checksum to ensure they have not been tampered with. Such checksums would also be supplied to Ironic with a path or url to the file. Supplying an allowlist is required in order to perform attestation. Excludelists are not required but are used in a majority of Keylime use cases.

The paths of the allowlist, checksum, and excludelist can be saved in `driver_info\keylime_allowlist`, `driver_info\keylime_allowlist_checksum`, and `driver_info\keylime_excludelist`.

Linuxs IMA submodule gathers measurement list signed with TPM quote. The Ironic controller will send the allowlist to the verifier using the Keylime tenant. The Keylime verifier obtains the measurement list and performs attestation by comparing the measurement list against allowlist.

Alternatives

We could add such functionality to various interfaces, but generally attestation will be a specific model for a deployment or portion of a deployment, and thus we may one day have need for vendor specific drivers for particular attestation solutions and workflow. As such, not creating a new interface for this seems less ideal.

Another alternative would be to perform certain checks along state transitions. For example, at clean time we can check the firmware and fail if things have been modified. However, this is undesirable in a scenario where we have strict workflows and processes we want to adhere to. In the situation where an owner lends a node to an untrustworthy lessee the owner might want to ensure the lessee does not perform any unexpected actions. This is also less extensible to other workflows such as a periodic monitoring.

Data model impact

Addition of a `attestation_interface` field to the node object, and this will require a database migration to create the field. The field will default to `None` which will map to a no-attestation interface.

State Machine Impact

No impact to the state machine is expected. All calls to the new interfaces methods will take place in existing workflows driven by the state machine. Action will be taken on a result immediately upon receiving the result.

REST API impact

The `attestation_interface` will be added to the node object and guarded by an API microversion.

Client (CLI) impact

ironic CLI

None

openstack baremetal CLI

The OSC plugin will be changed accordingly to assist users in changing the new `attestation_interface` field.

RPC API impact

This new `attestation_interface` field requires the RPC version to be incremented.

Driver API impact

The attestation interface methods that would be proposed would consist of a `no-attestation` interface defined on a new base class AttestationInterface.

These methods would consist of:

```
def validate_security_status(self, task):
    """Grabs the latest information about the node's security state
    from the attestation machine. Returns nothing on success, raises
    an exception if status is not what we expect or unable to reach
    verifier to obtain a status.
    """

def start_attestation(self, task):
    """Grabs the allowist, allowlist checksum, and excludelist from
    ``driver_info`` instructions. Verifies the integrity of the allowlist
    using the checksum. Attempts to send the allowlist and excludelist to
    the attestation service. Sending allowlist and excludelist allows the
    node to begin attesting itself. Returns nothing on success, raises an
    exception if checksum does not pass or is unable to reach the
    verifier to send allowlist/excludelist.
    """

def unregister_node(self, task):
    """Unregisters the node from the verifier machine. Returns
    nothing on success, raises an exception if status is not what
    we expect.
    """
```

These methods can be used during the nodes cleaning and deployment time. The action taken on a particular security state will be configurable. Whether or not we raise an error on attestation failure will be configurable.

A few additional variables will need to be saved as part of `driver_info` in order to manage the node. These include:

`driver_info\keylime_allowlist` the allowlist for a node.

`driver_info\keylime_allowlist_checksum` a checksum for the allowlist to ensure the allowlist has not been tampered with.

`driver_info\keylime_excludelist` the excludelist for a node.

`driver_info\keylime_agent_uuid` the uuid for a Keylime agent. Needed for querying the verifier for a security status and associating an allowlist/excludelist pair with a node in the Keylime verifier.

Workflow

With all this in mind, we have devised the following workflow for deployment/ cleaning using a Keylime implementation of the attestation interface.

Beforehand, the operator will stand up a machine with the Keylime verifier and registrar. The user will generate their own allowlist, allowlist checksum, and excludelist for the node. An admin may make these files available on the same machine as the Ironic controller and pass in the filepath to `driver_info` or a non admin may make these files available to grab and instead pass in a url to `driver_info`. This step must be done before provisioning. The operator will also pass in how to locate the Keylime registrar and verifier to `driver_info.`

During the image building process the node image will be set up with an instance of the Keylime agent, as well as TPM, and IMA configurations which will allow the Keylime agent to run. The Keylime agent will register itself with the Keylime registrar automatically once started. At this point booting has begun and the node may send its first heartbeat back to the Ironic controller.

Next, start_attestation() will be called to send the allowlist and excludelist to the verifier. The conductor will make an rpc call to the agent to retrieve the Keylime agents uuid, the Keylime agents address, and the port which the Keylime agent is listening on. The Ironic controller will save these variables as `driver_info\keylime_agent_uuid`, `driver_info\keylime_agent_address`, and `driver_info\keylime_agent_port` for further use. If the conductor does not receive these credentials cleaning will fail.

The allowlist and excludelist will be sent to the verifier by calling the keylime_tenant cli programmatically. Once the verifier has received the allowlist and excludelist, attestation will begin. The verifier will periodically poll the Keylime agent for IMA measurements and compare them with the allowlist and excludelist to determine if the node has been tampered with. The verifier will record the status of the node, but take no action on the status.

At this point, the conductor may perform a validate_security_status() call to get the status of the node. If the status is what we expect, we may proceed. If the status is something we do not expect, or the controller is unable to access the verifier due to network issues, we will fail the deployment.

The Keylime agent will need to be unregistered with a call to unregister_node() to instruct the Keylime verifier to end its connection and remove the node from its database.

Here is a diagram for the anticipated workflow:

diagram { Image; Node; Keylime-tenant; Keylime-verifier; Keylime-registrar; activation = none; span_height = 1; edge_length = 250; default_note_color = white; default_fontsize = 12; Image -> Node [label = The node is booted with an image generated by diskimage-builder tool. Keylime and TPM environment is setup in the image]; Node -> Keylime-registrar [label = Makes a post request to register the Keylime agent on the node]; Keylime-registrar -> Node [label = Responses the node with an encrypted AIK]; Node -> Keylime-registrar [label = Makes an activation request with an ephemeral registrar key from TPM]; Keylime-registrar -> Node[label = 200 OK]; Node -> Keylime-tenant [label = First heartbeat]; Keylime-tenant -> Keylime-tenant [label = The allowlist and excludelist are provided by the user to the Keylime tenant command]; Keylime-tenant -> Keylime-verifier [label = Sends allowlist and excludelist and adds the Keylime agent uuid to the verifier]; Keylime-tenant -> Node [label =Gets TPM quote from the node to check the Keylime agents validity with the registrar]; Keylime-verifier -> Node [label =Starts polling the node for verification]; Keylime-tenant -> Keylime-verifier [label = Gets the current status of the node]; }

Workflows which allow node lessees to bring their own Keylime instance in to attest a node is theoretically possible within the framework given in this spec. However, Keylime currently lacks certain features needed to make this fully automated in Ironic.

Nova driver impact

None

Ramdisk impact

To have the Keylime agent work with TPM 2.0, certain libraries and configuration must be provided. These enhancements will come as part of the ramdisk. This includes tpm2-tss software stack, tpm2-tools utilities, and, although not required, the tpm2-abrmd resource manager.

Keylime-agent will be setup on the ramdisk. A new dib element will be created to install keylime-agent and make it run as a system service.

A new IPA extension will be needed to collect and send back to the conductor the keylime_agent_uuid, keylime_agent_address, and keylime_agent_port.

Security impact

It has a positive impact on security, since we can verify if the node is trustworthy by the attestation service.

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

The `attestation` interface will not be enabled by default, since the default will map to a `no-attestation` interface.

Config options

Options for configuring whether or not cleaning and deployment should fail in face of attestation failure will be part of the new `[keylime]` section

fail_clean_on_attestation_failure
: Boolean to determine whether to fail clean on attestation failure

fail_deploy_on_attestation_failure
: Boolean to determine whether to fail deploy on attestation failure

Developer impact

None

3.2.3 Implementation

Assignee(s)

Primary assignee:
Leo McGann <ljmcgann> lmcgann@redhat.com Danni Shi <sdanni> sdanni@redhat.com

Other contributors:
None

Work Items

- Add `attestation_interface` database field.
- Implement base interface addition
- Implement `no-attestation` interface.
- Add node RPC object field
- Add API support and microversion.
- Implement Keylime attestation interface.

3.2.4 Dependencies

None

3.2.5 Testing

Testing for this interface and basic functionality, as well as integration testing using the ansible-keylime-tpm-emulator for TPM emulation.

3.2.6 Upgrades and Backwards Compatibility

No issues are anticipated.

3.2.7 Documentation Impact

Documentation will be provided about how to use keylime-verifier and keylime-registrar.

3.2.8 References

https://github.com/keylime

3.3 Boot configuration API

https://bugs.launchpad.net/ironic/+bug/2044561

3.3.1 Problem description

Currently, an Ironic conductor generates various boot files (e.g. iPXE scripts) in the public directory of the HTTP server associated with it. Then dynamic DHCP configuration with Neutron is used to point a node at the right HTTP server.

For standalone installations, the DHCP configuration is usually static. It works well with a single conductor, but is not operational out-of-box in a multi-conductor setup: a node simply does not know which HTTP server to boot from.

3.3.2 Proposed change

New API will be added to Ironic to serve boot configuration files, with iPXE being the reference implementation. This approach will allow us to rely on the existing hash ring to direct the request to the right conductor. See *REST API impact* for further details.

This is how the boot process will work in the case of iPXE, standalone Ironic and static DHCP configuration:

1. The Nodes iPXE firmware initiates a new DHCP session.
2. The DHCP request reaches the Ironics DHCP server (usually dnsmasq).
3. The DHCP server responds with an IP address and the catch-all boot script. This boot script will be located next to an arbitrary conductor, quite likely the one co-existing with the DHCP server.
4. The boot script will direct the iPXE firmware to the URL in the form of `http://<IP>:6385/v1/boot/<MAC>/boot.ipxe` where <IP> is the IP of any Ironic API instance (most likely the one co-existing with the HTTP server), <MAC> is the nodes MAC address.
5. Ironic API will find the node by MAC and direct the request to the correct conductor.
6. The conductor responsible for the node returns the iPXE script.

Alternatives

One proposed approach is to enable coordination between conductors. While potentially useful in more cases, that proposal has a JSON RPC multiplication problem in a standalone setup.

The other approach to the problem is to introduce managed standalone DHCP. The first step towards it has already been done: Ironic can manage its dnsmasq server. In a multi-conductor setup, it may imply having several dnsmasq instances on the same network, which is a potentially problematic setup. The current implementation still requires some sort of coordination between conductor or a periodic task to disable access to the DHCP servers of unrelated conductors.

A much better, although also the most complex, option would be to use an existing DHCP server with API access. One such server is Kea. The main problem: such a complex change may be too much for Ironic right now. The contributors are spread thin already. On top of that, Ive been told that only paid addons give us what we need.

Data model impact

None

State Machine Impact

None

REST API impact

`GET /v1/boot/<MAC>/<NAME>`

<MAC>
: MAC address of any NIC of a node.

<NAME>
: Configuration name to request (e.g. `boot.ipxe`).

The API will find the node by the MAC address, check its provision state and call into the `get_boot_config` RPC on success. On failure, a generic HTTP 404 will be returned to avoid disclosing any further information.

Note

The whole API branch `/v1/boot` will not be versioned since we dont expect firmware implementations to support extra headers or any sort of reasonable version negotiation.

Client (CLI) impact

None. The API is not for end users.

openstack baremetal CLI

None

openstacksdk

None

RPC API impact

```
def get_boot_config(self, context, node_id, name):
    """Request a boot configuration file."""
```

Driver API impact

The `boot` interface will get a new call:

```
def get_boot_config(self, task, name):
    """Request a boot configuration file.

    :param task: a TaskManage instance with a shared lock.
    :param name: configuration name.
    :raises: NotFound if the configuration cannot be produced.
    """
    raise exception.UnsupportedDriverExtension(
        driver=task.node.driver, extension='get_boot_config')
```

A reference implementation will be added to the iPXE boot interface, supporting a configuration called `boot.ipxe` - the iPXE script.

Nova driver impact

None

Ramdisk impact

None

Security impact

The new API will allow enumerating nodes in certain states by their MAC addresses. Some information may potentially be exposed by the boot configuration. The enumeration is already possible with the lookup API, and the configuration can be leaked by the boot scripts in the HTTP server. We will advise operators to limit access to the new API endpoints.

On top of that, the boot interfaces `validate` will not be called to avoid exposing information about the node fields.

Other end user impact

None

Scalability impact

Serving boot scripts via the API is somewhat less efficient than from an HTTP server. Operators concerned about the impact can opt into using the old approach.

We will avoid using an exclusive lock or launching additional threads in the implementation. The initial version will just read the existing files from disk.

Performance Impact

None

Other deployer impact

The feature will be configured with a few new options:

`[pxe]ipxe_use_boot_config_api = False`
: Enables the feature. If true, the generated root iPXE script (`boot.ipxe`) will contain links to the boot configuration API, not to the HTTP server.

`[pxe]ipxe_config_api_root_url = <None>`
: Specifies the root URL to use for links to the boot configuration API. An example use case is an Ironic deployment with TLS: iPXE does not support custom certificates without recompiling the firmware, so e.g. a proxy must be established instead.

`[api]restrict_boot_config = True`
: Instructs the API to be restricted to only nodes in * `WAIT` states. Operators using fast-track may want to set this to False.

Developer impact

None

3.3.3 Implementation

Assignee(s)

Primary assignee:
: Dmitry Tantsur (dtantsur)

Work Items

TODO

3.3.4 Dependencies

None

3.3.5 Testing

We can switch Bifrost to the new API. Its highly likely that Metal3 will also switch to it in the near future as we develop its HA story.

3.3.6 Upgrades and Backwards Compatibility

None.

3.3.7 Documentation Impact

Installation guide may need to be adjusted.

3.3.8 References

3.4 Dell EMC hardware firmware update

https://storyboard.openstack.org/#!/story/2003494

Operators and deployers (such as tripleo) need the ability to flash the firmware on Dell EMC hardware to specific versions before provisioning baremetal servers.

3.4.1 Problem description

Use cases

- An operator upgrades the firmware for certain hardware components on a server to newer versions to take advantage of new features or bug fixes in the firmware. This could be prior to or after provisioning.
- An operator rolls back the firmware for certain hardware components on a server to prior versions to avoid regressions introduced in newer firmware. This could be prior to or after provisioning.
- A deployer (such as tripleo) pins the firmware for certain hardware components to specified versions prior to initiating overcloud deployment.

The following use cases are considered outside the scope of this spec:

- An operator or software component uploads a firmware image to a firmware image repository.
- An operator or software component removes a firmware image from a firmware image repository.

Dell EMC WSMAN firmware management

WSMAN firmware management is offered by the iDRAC. Supported operations are:

- List firmware on server: Enumerate DCIM_SoftwareIdentity
- Update firmware: Invoke DCIM_SoftwareInstallationService.InstallFromSoftwareIdentity

Dell EMC Redfish firmware management

Redfish firmware management is offered by the iDRAC. Supported operations are:

- List firmware on server: https://\protect\T1\textdollaridrac_ip/redfish/v1/UpdateService/FirmwareInventory
- Update firmware: https://\protect\T1\textdollaridrac_ip/redfish/v1/UpdateService/Actions/UpdateService.SimpleUpdate

Support for firmware management in `redfish` hardware type

A patch for firmware update in sushy has been merged: https://review.opendev.org/#/c/613828/ There is no support for firmware update in the `redfish` hardware type yet.

3.4.2 Proposed change

Since the Dell EMC redfish implementation of firmware update is fully compliant with the DMTF spec, the decision has been made to go with a `redfish` implementation.

A manual clean step will be added to the RedfishManagement class to initiate firmware update. The clean step will accept a list of dictionaries. Each dictionary will represent a single firmware update, and will contain a URI to the firmware image.

As an example:

```
"clean_steps": [{
    "interface": "management",
    "step": "update_firmware",
    "args": {
        "firmware_images":[
            {
                "url": "file:///firmware_images/idrac/9/iDRAC-with-Lifecycle-
↪Controller_Firmware_VRYKT_WN64_3.32.32.32_A00.EXE",
                "checksum": "<sha1-checksum-of-this-file>"
            },
            {
                "url": "swift://firmware_container/BIOS_W8Y0W_WN64_2.1.7.EXE",
                "checksum": "<sha1-checksum-of-this-file>"
            }
        ]
    }
}]
```

The implementation will apply the firmware updates in the given order.

The implementation will have no knowledge of dependencies of the supplied firmware, or if the firmware is applicable to the hardware that it is being installed on. The implementation will rely on the firmware update failing gracefully in these cases.

The updater will fail fast so that if one update fails, it will abort and not apply the remaining updates. If a failure does occur midway through applying the updates, successful updates prior to the failed update will not be rolled back.

The cleaning step will be out-of-band. The firmware update cleaning step will use Redfish to perform the update. The intent is to use the sushy library if possible, and if not, provide vendor extensions as

necessary.

While the iDRAC supports rolling back to the last known good firmware, the ability to do this will not be implemented as part of this spec. Instead, if a user wishes to roll back to an early version of the firmware, they will just do a firmware update to an older version.

While the initial implementation of this will use the Redfish protocol, it will be implemented in such a way that it will not preclude adding support for the `WSMAN` protocol at a later date.

Alternatives

One alternative would be to implement firmware update using `WSMAN`. Because `WSMAN` will eventually be deprecated in favor of Redfish, it is preferred to avoid this option.

Another alternative would be to do firmware update in-band via an Ironic Python Agent hardware manager for the iDRAC.

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

ironic CLI

Users will be able to launch a cleaning step to update the firmware on Dell EMC servers.

openstack baremetal CLI

Users will be able to launch a cleaning step to update the firmware on Dell EMC servers.

RPC API impact

None

Driver API impact

A cleaning step will be added to update the firmware on Dell EMC hardware managed by the `redfish` hardware type.

Nova driver impact

None

Ramdisk impact

None

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

To use firmware update, the user will need to configure the selected hardware type to use the `redfish` management interface and set `redfish` credentials in the nodes driver_info.

Config options

A new [firmware_update] group will be defined in the ironic configuration file. The following options will be moved from the iLO section to that group.

use_web_server_for_images
: Indicates if images should be uploaded to the conductor web server.

swift_container
: The swift container for firmware images.

swift_object_expiry_timeout
: The timeout in seconds after which the given swift URL should expire.

Developer impact

None

3.4.3 Implementation

Assignee(s)

cdearborn

Work Items

- Ironic RedfishManagement changes to add a cleaning step
- python-dracclient changes to implement firmware update
- Ironic iDRAC hardware type changes to add support for the Redfish management interface

3.4.4 Dependencies

None

3.4.5 Testing

Addition of unit tests to test the firmware update cleaning step.

3.4.6 Upgrades and Backwards Compatibility

If a firmware update is attempted on a Dell EMC server that does not support the Redfish UpdateService.SimpleUpdate firmware upgrade command, then cleaning will be aborted and an appropriate error message logged.

3.4.7 Documentation Impact

The documentation will be updated to cover this new feature. The documentation will be updated to include the generations of Dell EMC hardware officially supported.

3.4.8 References

- Manual cleaning - https://github.com/openstack/ironic-specs/blob/master/specs/approved/manual-cleaning.rst
- sushy - https://github.com/openstack/sushy

3.5 API Evolution: etag ID

As time has gone on, the API surface has evolved, and the community has given rise to more use-cases and more users. This has resulted in the identification of several shortcomings in the current REST API: it is not as user-friendly as one might expect, and it does not service all of the use-cases we wish that it would.

This specification describes the implementation of etag identifiers on API responses, allowing for better conflict resolution between concurrent API clients.

https://bugs.launchpad.net/ironic/+bug/1605728

3.5.1 Problem description

Multiple clients attempting to update the same resource can overwrite each others changes accidentally and without notice. This is referred to as the lost update problem, and it is a problem in the Bare Metal service. The lost update problem is commonly addressed through the use of etag identifiers, as described in this API Working Group specification on etags. This specification proposes that the Bare Metal service provides etags support to address this problem.

3.5.2 Proposed change

The Bare Metal service shall begin to store, internally, a unique etag identifier for every API modifiable resource (Node, Port, Portgroup, Chassis). This identifier shall be returned in the body and headers of successful responses to GET requests for requested resource beginning with the introduction of a new microversion. There are mainly two cases:

- GET one resource(subresource). The ETag will be returned in the response headers and body. Python clients may operate with resource using fresh etag (see client section). Ironic shell users see etag in the response.

- GET list of resources(subresources). Etag of each resource will be available in response body, as field of the resource:

  {

  ports: [

  {
  uuid: 11111111-2222-3333-4444-555555555555,
  <all other fields>,
  etag: W/eeeeeeeeeeeee
  },
  {
  uuid: 66666666-7777-8888-9999-222222222222,
  <all other fields>,
  etag: W/ttttttttttttt
  }

  ]

  }

  Note that, putting etags to headers does not make sense, because it is impossible to distinguish etags at client side in standard and simple way.

All requests to modify a resource or subresource, whether through a PUT, PATCH, or DELETE request, SHOULD begin to require If-Match header with an appropriate etag identifier to be supplied in the request (for subresource it is etag of that subresource). Note that If-Match will not be required according to RFC standard specification of If-Match Header.

Therefore SHOULD keyword used here which means that `If-Match` header will be optional parameter in client (see keywords to indicate requirement levels).

This is important that originally according to rfc entity-tags using `If-Match` MUST be provided by clients to pass request successfully. When ETags are used for cache validation it is useful. Specifically to our use case making `If-Match` header necessary will decrease user-friendliness of ironic client. It is up to users to decide if they want to be aware of what they change.

`If-None-Match` or any other Precondition Header Fields will not be supported.

To save efficiency of the request the If-Match header SHALL be validated at API and conductor side by comparing the supplied If-Match header against the current etag string. If the supplied header does not match the actual value, a `412 Precondition Failed` SHALL be returned.

On the successful receipt of any request to modify a resource or subresource, a new etag identifier SHALL be generated by the server (it MUST NOT depend on which `ironic-api-version` came in the request). Where the modification is synchronous and the response already includes a representation of the resource, the new etag identifier SHALL be included in the response body and header.

Etag is a SHA-512 hash generated from JSON string encoded compactly (ordering of dict is not needed) from the dictionary of oslo versioned object fields. Etag will be generated for ALL create and modify requests taking into account the fields except of the ignored ones:

For node ignored fields are:

`driver_internal_info, etag, updated_at`

For port, portgroup and chassis it is only `etag` and `updated_at`.

The generated etag will contain W/ prefix to specify that weak validator is used. Strong validators are more useful for comparison, but in our use-case, taking into account, that etag is not changed on every update and not when metadata changes (e.g. Content-Type), weak validators are applied. See weak vs strong validators.

Alternatives

If we do not implement etag support, we will not have any means to prevent races between clients PATCH requests, and we will not be able to implement support for PUT as a means to update resources or sub-resources.

Data model impact

As on every REST API request the objects are obtained from database and while it is not going to be changed, etag shall be retrieved from object returned from db. This is more efficiently than spending time to generate hash again and again.

For that purpose a new field `etag` SHALL be added to each resource table to store the etag identifier. Etag identifier will be String Field with data length limited to 130 characters (etag is a string containing prefix W/ + SHA-512 hexadecimal number 128 digits long).

New field `etag` SHALL be also added to Object model in order to be consistent with db layer. Object layer will also be the place where etag will be regenerated based on current Object fields.

Etag will be also included to notifications payload to make them more flexible and usable.

State Machine Impact

None

REST API impact

A new etag header MUST be sent in response to all GET and POST requests starting from specific API microversion, as well as synchronous PATCH and PUT requests.

The same If-Match etag header SHOULD be accepted in all PUT, PATCH and DELETE requests. That means that each endpoint offering any update capability should have logic to validate etag optioanlly.

A new error status `412 PreconditionFailed` SHALL be introduced and used to signal the clients that their version of a resource is stale, when the supplied etag header does not match the servers version.

Client (CLI) impact

Using new microversion clients get the ability to update resource being aware of what exactly they change. For that they SHOULD send an `If-Match` header with an etag identifier and supported Ironic API version in the header of requests to modify any resource. There are two options: doing this either through CLI or through Python Client API. The last option is available for any python developer script used in clouds (useful for production).

The workflow of etags usage in ironicclient shell:

- Client does GET request.

- Starting from specific API microversion, response SHALL include an etag for requested resource(s) in the headers. ETag SHALL be also included within returned resources in the body, both for GET individual resource and resource collection.
- Etag may be specified by users if needed when doing any operation leading to resource changing by adding `--etag` flag to the command. Etag can be obtained from body or headers of response:

```
ironic --ironic-api-version 1.40 node-update \
--etag <etag_string> driver_info/foo=value
```

  This etag string is put as `If-Match` header to the request.

- Ironic API pops `If-Match` header, checks it with rpc_nodes etag and if they match, the entity tag is sent further to RPC where conductor validates it again. If etags are not matching at some point, the `412 PreconditionFailed` error will be raised. If requested `X-Openstack-Ironic-Api-Version` does not support etags yet, `NotAcceptable` error is raised.

To make Python Client API usable without shell, resources will be stored as full-featured objects (not just the bag of attributes), including the etag identifier. To do this the ironicclient API will be rewritten in the way that the `Resource` class is able to update itself and call manager to send requests available in NodeManager. The `Resource` will be stored in the memory like all Python objects are stored during process execution.

For the Python API all appropriate actions of `Resource` object will accept optional parameter etag. The workflow can be the following:

- In Python Shell or in some script clients do GET request to the resource. The etag returned in the response will be stored in the resource representation. E.g.

  `node = node_manager.get(node_ident)`

- Afterwards at any time user scripts can do the action on the resource itself:

  `new_node = node.update(patch, etag=True)`

  Afterwards they will have the up-to-date resource representation if the request will be validated on the server.

  Note, that for 1 standard deprecation period `If-Match` will not be sent to server by default. Clients will be warned that in the next release `etag` parameter will be `True` by default.

If etag is requested it will be retrieved from current resource representation

(as `node.etag` or `getattr(node, 'etag')`). Afterwards it will be sent as `If-Match` header, it means that user cares about up-to-date information. If etag is not present at the resource and clients did not turn off `etag` option, they will fail if using the API version greater or equal than etag API version.

Depending on the situation, the client may choose to transparently retry, or display a diff to the user of the stored vs. server-side resource. Clients should also begin alerting the user when an update request fails due to a resource conflict.

ironic CLI

See above.

openstack baremetal CLI

Same workflow described in the Client Impact.

RPC API impact

RPC API version needs to be bumped to accept etag parameter for actions on the resource. The `etag` parameter, default as None, should be passed to the appropriate methods.

Driver API impact

None

Nova driver impact

Nova ironic driver may use new Ironic API microversion, so ironic api version used by nova virt driver needs to be bumped. Until etag option in python ironicclient API is True, in the nova driver we should explicitly specify `etag=False` in appropriate methods through `Node` resource object.

Ramdisk impact

None

Security impact

None

Other end user impact

Sending `If-Match` header may fail due to `412 Precondition Failed` error. A client may retry with new fresh etag or/and display a diff between two resources representations.

Scalability impact

None

Performance Impact

New etags generation may increase the time to respond depending on the resource size.

Other deployer impact

Some services (e.g. Nova) change baremetal resources through API, so they may upgrade Ironic API to use etag. If services do not upgrade, warn the deployer about that in the case skipping these upgrades may violate some strong recommendations and information consistency is not guaranteed on ironic side.

Developer impact

Python developers can work with `Resource` object as with full-featured objects and perform modifying operations on them. Also they can implement scripts that are using etag option in parallel in efficient way.

3.5.3 Implementation

Assignee(s)

Primary assignee:
galyna

Other contributors:
vdrok

Work Items

- Implement database migration, adding an internal etag field to all top-level resources.
- Implement generation and validation utility functions in common code.
- Implement changes within the ironic.api.controllers.v1 modules to accept and return etag identifiers when fetching or changing resources as appropriate.
- Implement unit tests and tempest tests.
- Update api-ref documentation.
- Implement changes in the python client library and openstack CLI to begin caching etags on GET requests and sending etags on PUT/PATCH/DELETE requests.

3.5.4 Dependencies

None

3.5.5 Testing

Unit and tempest tests shall be added that ensure etag identifiers are returned, that they are validated by requests to modify resources, and that proper errors are returned when an invalid (or merely not current) etag is supplied.

3.5.6 Upgrades and Backwards Compatibility

Backwards compatibility is retained because etags SHOULD only be returned and required in new microversions.

This change does not include substantive changes to any resource.

3.5.7 Documentation Impact

The proper use of etag identifiers shall be documented in our API reference.

3.5.8 References

- https://tools.ietf.org/html/rfc7232
- http://specs.openstack.org/openstack/api-wg/guidelines/etags.html
- https://etherpad.openstack.org/p/ironic-v2-api

3.6 Expose supported power states

https://bugs.launchpad.net/ironic/+bug/1734827

This SPEC proposes adding a new API to expose supported power states of nodes. This API would be helpful to see if a power action is supported by a node because it depends on whether a power interface supports power actions such as soft power off and soft reboot.

3.6.1 Problem description

Ironic supports soft shutdown and soft reboot since Ocata. However, not all power interfaces support these new power actions. A new API would be necessary to see if a power action is supported by a node. This proposed API is similar to the one for getting a nodes supported boot devices, which is described here.

3.6.2 Proposed change

The new APIs will be introduced to get the current power status and the supported power states of nodes. See the REST API impact section and for the details:

- GET /v1/nodes/{node_ident}/states/power
- GET /v1/nodes/{node_ident}/states/power/supported

The new CLI will be also added for the new APIs. See the Client (CLI) impact section and for the details:

- `openstack baremetal node power show [--supported] <node>`

Alternatives

There is another option to only add a new API to get supported power states. There are also a few other options to support the API by the CLI:

- Add a new option to display the supported power states to `openstack baremetal node show`.
- Add a new subcommand to show the supported power states like: `openstack baremetal node supported power show`.

However, the current design is better for consistency with the existing APIs and the CLI for the boot devices.

Data model impact

None

State Machine Impact

None

REST API impact

We will introduce the two APIs. They will be available starting with a new Bare Metal API version.

- Get the current power state of a node:

  GET /v1/nodes/{node_ident}/states/power

  The response is like:

```
{
  "power_state": "power on"
}
```

- Get the supported power states of a node:

  GET /v1/nodes/{node_ident}/states/power/supported

  The response contains supported power states of the node: For example:

```
{
  "supported_power_states": [
    "power on",
    "power off",
    "rebooting",
    "soft rebooting",
    "soft power off"
  ]
}
```

Client (CLI) impact

The `openstack baremetal` CLI will support the new API.

ironic CLI

None

A new feature is no longer added to the ironic CLI.

openstack baremetal CLI

A new subcommand will be added:

```
openstack baremetal node power show [--supported] <node>
```

Without the `--supported` option, this command displays the power state of the specified node. When the `--supported` option is specified, this command displays the supported power states.

RPC API impact

A new RPC API, `get_supported_power_states` will be added. This returns a list of the supported power states of the specified node synchronously.

Driver API impact

None

The driver API `get_suuported_power_states` was already defined in the base power interface. If a power interface doesnt override the method, the default list which contains power on, power off, and reboot is returned.

Nova driver impact

None

The new API is available to see if a requested power action is supported or not. Though it might be helpful for the Nova driver, no change is planned currently.

Ramdisk impact

None

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

None

Developer impact

None

3.6.3 Implementation

Assignee(s)

Primary assignee:
shiina-hironori (irc:hshiina)

Other contributors:
None

Work Items

- Add the new RPC API.
- Add the new Baremetal APIs.
- Add the OSC baremetal subcommand.
- Add a new API test to tempest.
- Add the new APIs to the API reference.

3.6.4 Dependencies

None

3.6.5 Testing

An API test will be added to Tempest.

3.6.6 Upgrades and Backwards Compatibility

None

3.6.7 Documentation Impact

The new APIs will be added to the Baremetal API Reference.

3.6.8 References

This change was originally mentioned in reviewing a SPEC to support soft shutdown.: https://review.opendev.org/#/c/186700/

3.7 HTTPBoot

https://bugs.launchpad.net/ironic/+bug/2032380

A long sought after feature in Ironic is to boot a node from a single HTTP URL. Except, there is more than one flavor of this functionality.

A first flavor is very virtual media like where we ask the Baseboard management controller to boot the machine from a URL. The exact details are vendor specific, but in essence UEFI firmware boots the OS from the URL.

The second flavor is very similar to PXE booting using DHCP, however the primary difference is through the use of a vendor class of HTTPClient is utilized instead of PXEClient. The funny thing, IPv6 network boot requires the same DHCP response in the form of a URL for the bootfile-url.

The reason to note both is they are both *very* similar, and both can be tested in our CI at this point. Previously we deferred implementation of the DHCP path because a lack of ability to test that in CI, but that is no longer a constraint of late 2023.

Implementing both would be relatively easy and very beneficial for the operator ecosystem, while also allowing for the navigation of the security and NAT incompatibility issues which exist with TFTP.

3.7.1 Problem description

Infrastructure operators need reliable, and relatively secure means of conveying bootloaders and initial artifacts to remote machines.

Ironics historical answer has been to utilize virtual media.

But not all hardware supports virtual media, and emulating a block device is not exactly the simplest thing once you boot an operating system. A helpful aspect with UEFI and the evolution of systems is now we have facilities in Baseboard management controllers and even UEFI base firmware support retrieving initial boot payload from a remote system using HTTP(s).

3.7.2 Proposed change

Given the similarity, it makes sense to implement support for both the BMC oriented path and the DHCP oriented path as part of single effort.

The greater benefit for implementing the DHCP oriented path is we can also extend this functionality to our downstream consumers with minimal effort.

BMC Path

- Update sushy-tools to support mapping cals to utilize a HTTP next boot URL to the virtual media driver code. Functionally this is similar, as there appears to be no means to inject the hint to boot from the URL into libvirt.
- Update sushy to support requesting to boot a node from a HTTP URL.
- Create a new `redfish-http-url` boot interface, named `RedfishHTTPBoot` BootInterface class based upon the underlying class `RedfishVirtualMediaBoot`. In this class, replace `_insert_vmedia`, and `_eject_vmedia` class. In essence these methods would perform the needful calls to the BMC to perform actions such as setting `BootSourceOverrideEnabled` to Once, `BootSourceOverrideMode` to UEFI, `BootSourceOverrideTarget` to UefiHttp, and finally `HttpBootUri` to the ISO file we wish to boot.

Constraints:

- Limited to UEFI.

Note

We may wish to retool some of the internals of the RedfishVirtualMediaBoot class for our implementation sanity.

Note

This interface with the BMCs is modeled around use of an ISO image as a boot source, where as the DHCP path is modeled upon a bootloader, much like iPXE.

DHCP Path

- Determine if we need to modify the `neutron-dhcp-agent`.
- Create an `httpboot` BootInterface based upon the existing `pxe` interface code base, and wire through a flag which is set by the PXE utils invocation of the DHCP code base, to signal the use of an HTTP(S) URL to the conductor. Specifically it would take the form of a flag on the pxe_utils method `prepare_instance_pxe_config` which would be supplied to the `dhcp_options_for_instance` method call in the same file. Likely as simple as `pxe_base` looking for a feature flag on the BootInterface class.

Note

We may wish to make an iPXE specific version of the boot interface as well, which is *also* already handled by a capabilities feature flag. A separation would enable Grub and iPXE use concurrently.

Alternatives

We could limit scope, but there really are not any alternatives, and both paths provide a great deal of functionality and benefit to users of Ironic.

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

openstack baremetal CLI

None, this is entirely a server side capability/configuration.

openstacksdk

None, this is entirely a server side capability/configuration.

RPC API impact

None

Driver API impact

No changes to the Driver API are anticipated, although this change functionally proposes two or three different BootInterfaces to be created.

Nova driver impact

None

Ramdisk impact

None. Booting the ramdisk would take the existing code paths with slight deviation where applicable for each driver case.

Security impact

The overall security posture of deployments could improve with these capabilities. Specifically UEFI firmware can boot an ISO or first stage bootloader over HTTPS.

Other end user impact

None

Scalability impact

None anticipated.

Performance Impact

None anticipated.

Other deployer impact

Deployers interested in using this functionality will have expanded operational and security capabilities which are in-line with established interfaces and data models in Ironic.

Developer impact

None

3.7.3 Implementation

Assignee(s)

Primary assignee:
Julia (TheJulia) Kreger <juliaashleykreger@gmail.com>

Other contributors:
Volunteers welcome!

Work Items

- Add support to sushy and sushy-tools for the URL boot operation.
- Add support to pxe_utils logic to generate a URL boot response payload, and set that based upon a driver feature/capability flag.
- Compose lots of documentation.
- Create tempest suite to exercise both modes of boot operations.

3.7.4 Dependencies

IF we need to apply DHCP server configuration, similar to PXE/IPXE, chain loading attributes, then we will need to engage the Neutron developers.

3.7.5 Testing

The ideal path would be to create a single integration suite test in the ironic-tempest-plugin to set a node to utilize both interfaces, and toggle the nodes through a clean step, which would prove the interfaces work as we expect.

3.7.6 Upgrades and Backwards Compatibility

No issues are anticipated here.

3.7.7 Documentation Impact

We will likely need to compose more documentation than code in every case of these interface.

3.7.8 References

- https://www.dmtf.org/sites/default/files/standards/documents/DSP2053_2022.3.pdf
- https://bugs.launchpad.net/ironic/+bug/2032380
- https://en.opensuse.org/UEFI_HTTPBoot_Server_Setup

3.8 Migrate inspection rules from Inspector

https://storyboard.openstack.org/#!/story/2010275

This specification finishes the work started in /approved/merge-inspector by migrating the inspection rules API.

Please see the *Glossary* for the concepts required to understand this specification.

3.8.1 Problem description

Inspection is a pretty opinionated process. A few areas often require site-specific customizations:

- Auto-discovery. For example, credentials may be populated for new nodes following a certain pattern or by fetching them from a CMDB.
- Validation logic. Some operators want to fail inspection if the nodes do not fulfill certain criteria. In the context of auto-discovery, such a validation may be used to prevent unexpected machines from getting enrolled.

These requests can be covered by inspection hooks. But, as explained in *alternatives*, writing and deploying hooks can be too inflexible.

3.8.2 Proposed change

Migrate a streamlined version of introspection rules from Inspector.

These useful features are added on top of what Inspector provides:

Built-in rules
: Allow operators to load rules from a YAML file. These rules will be always present, wont be stored in the database and wont be deletable. Such rules are both an easier way to write hooks and a replacement for awkward Inspectors configuration options such as `[discovery]enabled_bmc_address_version`.

Phases
: In Inspector, rules are always run at the end of processing. Well add a new field `phase` to rules with values:

  - `early` - run before any other processing, even before auto-discovery and lookup. Such rules will not have access to the node object.
  - `preprocess` - run after the `preprocess` phase of all inspection hooks, but before the main phase.
  - `main` (the default) - run after all inspection hooks.

Updating rules
Inspector does not provide API for updating rules. There is no reason for that, and well add `PATCH` support for them.

Sensitive rules
Conditions and actions of the rules may contain sensitive information, e.g. BMC information. If a rule is marked as sensitive, its actions and conditions will not be returned as part of the `GET` request response. It will not be possible to make a sensitive rule not sensitive.

Error messages resulting from running sensitive rules will also be a bit terse to avoid accidentally disclosing sensitive information.

Priorities
In Inspector, rules are always run in the creation order. This is obviously inconvenient, so in Ironic well add priorities to them. Priorities between 0 and 9999 can be used by all rules, negative value and values above 10000 are reserved for built-in rules. The default priority is 0. Rules within the same priority are still run in the creation order for compatibility.

Database storage
Currently, Inspector spreads each rule into three tables (rules, conditions and actions). This may be more correct from the database design perspective, but is actually inconvenient to work with, since conditions and actions are never accessed outside of a Rule context. Nor are rules ever accessed without their conditions and actions. This specification buts them as JSON fields inside the Rule table.

Consistent arguments for conditions and actions
A condition has a `field` attribute that is special-cased to be a field of either the node or the inventory. This specification changes both to structure with one operation and several arguments, see *data model impact*.

Alternatives

- Use the inspection hooks mechanism. Hooks are less flexible since they require Python code to be installed alongside Ironic and the service to be restarted on any change. The former is especially problematic for container-based deployments.
- Radically change the rules DSL to something less awkward, e.g. Ansible-like miniscript. While I still want to do it eventually, I think such an undertaking will increate the scope of this already large work too much. With API versioning in place, we can always change the language under the hood.
- Allow API users to upload Python code. No comments.
- Seriously, though, write the rules in Lua or any other grown-up embedded language. I havent researched this option enough. Maybe its the way to go? Will deployers mind a new C dependency (on liblua or LuaJIT)?
- Copy inspection rules verbatim, no removals, no additions. I dont see why not make small improvements for better long-term maintenance. Some of the additions are related to security.
- Make inspection rules into a service separate from Ironic. Defeats the purpose of the Inspector merger. For example, no access to the node database means less efficient operations.
- Do not migrate inspection rules at all. They do bring a bit of complexity, but also proved a helpful instrument for operations. CERN uses them, which is my benchmark for usefulness among advanced operators.

Data model impact

Adapted from Inspector with the additions described in *Proposed change*:

```
class Rule(Base):
    uuid = Column(String(36), primary_key=True)
    created_at = Column(DateTime, nullable=False)
    updated_at = Column(DateTime, nullable=True)
    priority = Column(Integer, default=0)
    description = Column(String(255), nullable=True)
    scope = Column(String(255), nullable=True)  # indexed
    sensitive = Column(Boolean, default=False)
    phase = Column(String(16), nullable=True)  # indexed
    conditions = Column(db_types.JsonEncodedList(mysql_as_long=True))
    actions = Column(db_types.JsonEncodedList(mysql_as_long=True))
```

Conditions and actions

In this specification, both conditions and actions have the same base structure:

- `op` - operation: either boolean (conditions) or an action (actions).
- `args` - a list (in the sense of Python `*args`) or a dict (in the sense of Python `**kwargs`) with arguments.

The special attributes for actions in Inspector get a different form:

- Instead of `invert`: put an exclamation mark (with an optional space) before the `op`, e.g. `eq` - `!eq`.
- Instead of just `multiple`, support an Ansible-style `loop` field. On actions, several actions are run. On conditions, the `multiple` field defines how to join the results. Same as in Inspector:

  any (the default)
  : require any to match

  all
  : require all to match

  first
  : effectively, short-circuits the loop after the first iteration

  last
  : effectively, only runs the last iteration of the loop.

Variable interpolation

String arguments are processed by Python formatting with `node`, `ports`, `port_groups`, `inventory` and `plugin_data` objects available, e.g. `{node.driver_info[ipmi_address]}`, `{inventory[interfaces][0][mac_address]}`.

When running in the early phase, only `inventory` and `plugin_data` are available.

The `node` is actually a proxy mapping taking into account the `mask_secrets` option (described in *other deployer impact*).

If a value is a string surrounded by single curly brackets `{` and `}` (no unformatted text), well evaluate what is inside and avoid converting it into a string. This way, lists and dictionaries can be passed to actions and `loop`. This behavior will likely be implemented by hooking into the Formatter class.

Available conditions

Unlike in Inspector, a list of conditions will be built into Ironic:

is-true(value)
Check if value evaluates to boolean True. On top of actual booleans, non-zero numbers and strings yes, true (in any case) are evaluated to True.

is-false(value)
Check if value evaluates to boolean False. On top of actual booleans, zero `None` and strings no, false (in any case) are evaluated to False.

> **Note**
>
> These conditions can both be false for some values (e.g. random strings). This is intentional.

is-none(value)
Check if value is None.

is-empty(value)
Check if value is None or an empty string, list or a dictionary.

eq/lt/gt(*values, *, force_strings=False)
Check if all values are equal/less than/greater than. If `force_strings`, all values will be converted to strings first.

> **Note**
>
> Inspector has `ne`, `le` and `ge`, which can be implemented via `!eq`, `!gt` and `!lt` instead.

in-net(address, subnet)
Check if the given address is in the provided subnet.

contains(value, regex)
Check if the value contains the given regular expression.

matches(value, regex)
Check if the value fully matches the given regular expression.

one-of(value, values)
Check if the value is in the provided list. Similar to `contains`, but also works for non-string values. Equivalent to:

```
- op: eq
  args: [<value>, "{item}"]
  loop: <values>
```

Available actions

Similar to Inspector, actions will be plugins from the entry point `ironic.inspection_rules.actions`. Coming with Ironic are:

fail(msg)
Fail inspection with the given message.

set-plugin-data(path, value)
Set a value in the plugin data.

extend-plugin-data(path, value, *, unique=False)
Treat a value in the plugin data as a list, append to it. If `unique` is True, do not append if the item exists.

unset-plugin-data(path)
Unset a value in the plugin data.

log(msg, level="info")
Write the message to the Ironic logs.

The following actions are not available in the `early` phase:

set-attribute(path, value)
Set the given path (in the sense of JSON patch) to the value.

extend-attribute(path, value, *, unique=False)
Treat the given path as a list, append to it.

del-attribute(path)
Unset the given path. Fails on invalid node attributes, but does not fail on missing subdict fields.

set-port-attribute(port_id, path, value)
Set value on the port identified by a MAC or a UUID.

extend-port-attribute(port_id, path, value, *, unique=False)
Treat the given path on the port as a list, append to it.

del-port-attribute(port_id, path)
Unset value on the port identified by a MAC or a UUID.

> **Note**
>
> Here *path* is a path in the sense of a JSON patch used by Ironic API.

Examples

Partly taked from the Inspector docs, using YAML format.

```
- description: Initialize freshly discovered nodes
  sensitive: true
  conditions:
    - op: is-true
      args: ["{node.auto_discovered}"]
    - op: "!is-empty"
      args: ["{plugin_data[bmc_address}"]
  actions:
    - op: set-attribute
      args: ["/driver", "ipmi"]
    - op: set-attribute
      args: ["/driver_info/ipmi_address", "{plugin_data[bmc_address]}"]
    - op: set-attribute
      args: ["/driver_info/ipmi_username", "admin"]
```

(continues on next page)

(continued from previous page)

```
    - op: set-attribute
      args: ["/driver_info/ipmi_password", "pa$$w0rd"]
```

Note

The `plugin_data[bmc_address]` field is a side-effect of the `validate_interfaces` hook.

```
- description: Initialize Dell nodes using IPv6
  sensitive: true
  conditions:
    - op: is-true
      args: ["{node.auto_discovered}"]
    - op: contains
      args: ["{inventory[system_vendor][manufacturer]}", "(?i)dell"]
  actions:
    - op: set-attribute
      args: ["/driver", "idrac"]
    - op: set-attribute
      args: ["/driver_info/redfish_address", "https://{inventory[bmc_
↪v6address]}"]
    - op: set-attribute
      args: ["/driver_info/redfish_username", "root"]
    - op: set-attribute
      args: ["/driver_info/redfish_password", "calvin"]
```

State Machine Impact

None (rules are running in the `INSPECTING` state)

REST API impact

Migrate the API mostly verbatim, changing the prefix to `inspection_rules`, adding `PATCH` and more options for listing:

`POST /v1/inspection_rules`
Create an inspection rule. The request body is the representation of the rule. All fields, except for `built_in` can be set on creation. Only `actions` are required (rules with empty conditions run unconditionally).

Returns HTTP 400 on invalid input.

`GET /v1/inspection_rules/<uuid>`
Return one inspection rule. The output fields mostly repeat the database fields, adding a boolean `built_in` field.

For sensitive rules, `null` is returned instead of `conditions` and `actions`.

Returns HTTP 404 if the rule is not found.

`GET /v1/inspection_rules[?detail=true/false&scope=...&phase=...]`
List all inspection rules. If `detail` is `false` or omitted, conditions and actions are not returned. Filtering by scope and phase is possible.

Returns HTTP 400 on invalid input.

PATCH /v1/inspection_rules/<uuid>
Update one rule and return it. Sensitive rules can be updated, but the result does not contain conditions or actions in any case.

Returns HTTP 404 if the rule is not found.

Returns HTTP 400 if the input is invalid, e.g. trying to modify `built_in`, change `sensitive` to `false` or set priority outside of the allowed range (0 to 9999).

DELETE /v1/inspection_rules/<uuid>
Remove one rule.

Returns HTTP 404 if the rule is not found.

Returns HTTP 400 if the rule is built-in.

DELETE /v1/inspection_rules
Remove all rules except for built-in ones.

Client (CLI) impact

openstack baremetal CLI

Inspection rules CRUD, adapted from the Introspection Rules CLI, simply replacing *introspection* with *inspection*:

```
$ openstack baremetal inspection rule import <file>
$ openstack baremetal inspection rule list [--long]
$ openstack baremetal inspection rule get <rule ID>
$ openstack baremetal inspection rule delete <rule ID>
```

The mass-deletion command is changed for clarity:

```
$ # Inspector version:
$ openstack baremetal introspection rule purge
$ # New version:
$ openstack baremetal inspection rule delete --all
```

Updating will be possible:

```
$ openstack baremetal inspection rule set <rule ID> \
      [--actions '<JSON>'] [--conditions '<JSON>'] \
      [--sensitive] [--scope '<scope>'] [--phase 'early|preprocess|main'] \
      [--uuid '<uuid>'] [--description '<description>']
$ openstack baremetal inspection rule unset <rule ID> \
      [--conditions] [--scope] [--description]
```

Also adding a way to create from fields instead of one JSON:

```
$ openstack baremetal inspection rule create \
      --actions '<JSON>' [--conditions '<JSON>'] \
      [--sensitive] [--scope '<scope>'] [--phase 'early|preprocess|main'] \
      [--uuid '<uuid>'] [--description '<description>']
```

openstacksdk

The baremetal module will be updated with the standard CRUD plus mass-deletion:

```
def inspection_rules(details=False): pass
def get_inspection_rule(rule): pass
def patch_inspection_rule(rule, patch): pass
def update_inspection_rule(rule, **fields): pass
def delete_inspection_rule(rule, ignore_missing=True):
def delete_all_inspection_rules(): pass
```

RPC API impact

None

Driver API impact

No driver impact. Operators may opt for running inspection rules on nodes with all inspect interfaces, including out-of-band ones.

Nova driver impact

None

Ramdisk impact

None

Security impact

Inspection rules have access to all node and inventory data. Thus, they should be restricted to admins only.

Other end user impact

None

Scalability impact

None

Performance Impact

Having a lot of inspection rules will make inspection longer. But it should not affect the rest of the system.

Other deployer impact

The new section `[inspection_rules]` will have these options:

`built_in`
: An optional path to a YAML file with built-in inspection rules. Loaded on service start and thus not modifiable via SIGHUP.

`default_scope`
: The default value for `scope` for all rules where this field is not set (excluding built-in ones).

mask_secrets
: Whether to mask secrets in the node information passed to the rules:

  - `always` (the default) - always remove things like BMC passwords.
  - `never` - never mask anything, pass full node objects to all rules.
  - `sensitive` - allow secrets for rules marked as `sensitive`.

supported_interfaces
: A regular expression to match *inspect interfaces* that run inspection rules. Defaults to `^(agent|inspector)$` to limit the rules to only in-band implementations. Can be set to `.*` to also run on all nodes.

One option will be added to the `[auto_discovery]` section:

inspection_scope
: The default value of inspection scope for nodes enrolled via auto-discovery. Simplifies targeting such nodes with inspection rules.

Developer impact

Actions are provided via plugins with entry points in the `ironic.inspection_rules.actions` namespace:

```
class InspectionRuleActionBase(metaclass=abc.ABCMeta):
    """Abstract base class for rule action plugins."""

    formatted_params = []
    """List of params to be formatted with python format."""

    supports_early = False
    """Whether the action is supported in the early phase."""

    def call_early(self, rule, *args, **kwargs):
        """Run action in the early phase."""
        raise NotImplementedError

    @abc.abstractmethod
    def __call__(self, task, rule, *args, **kwargs):
        """Run action on successful rule match."""
```

Note

The interface in Inspector supports several additional validation features. I hope to derive the valid arguments from method signatures instead.

3.8.3 Implementation

Assignee(s)

Primary assignee:
: Dmitry Tantsur (IRC: dtantsur, dtantsur@protonmail.com)

Other contributors:
TBD

Work Items

See the RFE.

3.8.4 Dependencies

- /approved/merge-inspector

3.8.5 Testing

- Add functional tests exercising inspection rules CRUD actions.
- Update the in-band inspection job to have a simple rule that we can verify is run (e.g. it sets something in the nodes extra).

3.8.6 Upgrades and Backwards Compatibility

Existing rules will not be automatically migrated from Inspector to Ironic since the conversion may not be always trivial (e.g. around variable interpolation or loops).

3.8.7 Documentation Impact

- API reference will be updated.
- User guide will be migrated from Inspector with a couple of real-life *examples*.

3.8.8 References

3.9 Redfish Proxy for Ironic Node Power Controls

https://storyboard.openstack.org/#!/story/2009184

One important aspect of our goal of improving support for node multi-tenancy is to provide a seamless user experience for node lessees. This includes enabling lessees to use their existing provisioning workflows on leased nodes and to have these tools work as expected. Ironic policy does allow for lessees to access basic power controls through either the Ironic REST API or the `openstack baremetal` CLI; however, if such tools lack instructions for communicating with the Ironic API, the only way for them to function is by making requests directly to the BMC of the node they wish to provision. This would require giving the lessee BMC power credentials, which is especially problematic considering that one of the goals of node multi-tenancy is to enable this sort of functionality without giving the user such low-level access. [NMTSTORY] [NODELEAS]

Because such a solution is undesirable, to enable the use of existing provisioning tools, we instead intend to emulate a standards-based interface (Redfish, in this case) for use by node lessees. This interface will receive Redfish API calls such as those made by provisioning tools, perform the corresponding action within Ironic, and return a response in the format defined by the Redfish schema. Redfish clients will be able to authenticate in the way they expect to and these requests will be handled using the same authentication strategy as the rest of the existing Ironic infrastructure. If all goes according to plan, this feature will solve the compatibility problem previously described, and will do so without giving any special permissions to the lessee.

3.9.1 Problem description

The main concern here is compatibility the problem in which a lessee has tools that expect to communicate via an inaccessible interface has two undesirable solutions.

The first is for the lessees workflow to be refactored to instead make use of the Ironic API. For the user this is both time and resource-consuming, especially considering these lessees will often have access to bare metal nodes for a limited timeframe.

The second is to provide the lessee with BMC credentials, which aside from going against the principles of node multi-tenancy, is a huge security risk. If these credentials are not limited in access scope, lessees could possibly damage or otherwise compromise the bare metal node. Additionally, if the provided credentials are not immediately revoked when the lessee is supposed to lose access to the node, they would retain access to the nodes power controls (and potentially more) until the credentials are changed.

These solutions potentially address the compatibility issue, but in turn, they create either a major efficiency issue or a major security issue. This feature intends to address compatibility without significantly compromising security or efficiency.

3.9.2 Proposed change

We will create a new WSGI service within the main Ironic repo which will serve a REST API at `/redfish` that to the end user, will function like a legitimate v1 Redfish endpoint, albeit with a limited feature set. This interface will provide minimal functionality, just enough to allow Redfish provisioning tools to set the power state of an Ironic node. In the future, this service could be extended to provide additional features, such as boot devices, boot modes, and virtual media; however, this is beyond the scope of what we aim to achieve in this spec.

Implementation details

The three key components necessary to achieve this goal are:

1. a means by which users can be authenticated
2. a way to determine the users identity (roles, projects, privileges, etc.)
3. interfaces for performing operations on the nodes the user wishes to provision.

Broadly, the workflow for making use of this feature shall be as follows:

1. Acquire credentials to be used for authenticating with the Ironic Redfish intermediary.
    - If the Ironic system in question uses Keystone for authentication, the user must create a set of application credentials scoped to the appropriate project(s) and domain.
    - If not, the user will authenticate via HTTP basic authentication, which will be handled using Ironics basic auth middleware. Configuring the backend where credentials are stored will be left up to individual Ironic system operators.
2. (Keystone users only) Authenticate using the Redfish SessionService interface.
    - Keystone users will authenticate through the Redfish SessionService interface using the UUID of their newly created application credential as a username and the credentials secret as a password. In response, they will receive a Redfish schema-compliant Session object in the body, as well as an authorization token and the URL of the newly created Sessions interface in the header.
3. Perform the necessary provisioning operations.

- All requests to Systems endpoints will require authentication; users who authenticated with the SessionService in step 2 must provide the `X-Auth-Token` they received in each request header, and users working with HTTP Basic Auth must provide base64-encoded credentials in each request header (as specified in [RFC7617]).

4. (Keystone users only) End the Session created in step 2.

   - Keystone users will send a DELETE request to the URL of the Session object returned to them previously, internally revoking the created Keystone authentication token (note: not the application credential). If the user wishes to perform further actions, they will need to repeat the authentication process from step 2 again.

Authentication

The actions that this feature shall provide access to should only be accessed by certain users. The question of authentication is one of is the person that is requesting access really who they claim to be? How we are to answer this question depends on the authenticaton strategy in place on the Ironic system in question.

In the context of Keystone (v3), the question of who is this person? requires a few pieces of information the users identifier, the identifier of the project theyre working on, and the identifier of the domain in which the user and project exist. However, Redfish expects only the identifier of the user (UUID or username) to determine identity and as such, authentication would end up requiring the Redfish user to provide more information than they would otherwise expect to provide, which poses a potential problem.

We intend to solve this problem through the use of application credentials, which specify the user, project, and domain they are scoped to. Since each application credential possesses a UUID, we can use this identifier in place of all the information that would otherwise be required by Keystone. [APPCREDS] This approach is also beneficial from a security standpoint, as it reduces the number of times raw user credentials are handled directly.

The user will pass the credentials UUID and secret to the SessionService, where it will internally be passed along to Keystone for verification. If the information provided is valid, an authorization token will be created and sent back to the user in a format emulating that of a Redfish Session. Since Redfish Sessions are required to have a UUID, we will use the audit ID of the newly created Keystone auth token to satisfy this requirement. According to the Keystone API reference, You can use these audit IDs to track the use of a token without exposing the token ID to non-privileged users. [KSTNEAPI] We want to make sure this proxy does not unintentionally expose sensitive information, and the decision to use audit IDs seems like a sensible one in the context of this problem.

Once the user is finished performing the provisioning actions they intend to carry out, they will send a DELETE request to the Session URL, as per the Redfish spec. Internally, this will revoke the Keystone authorization token, essentially logging the user out. This is the intended method for ending a user session, however it is important to note the difference between how Keystone and Redfish handle session expiration.

Redfish Sessions are designed to expire after a period of inactivity, while Keystone authorization tokens are designed to expire at a specific time (e.g. an hour or two after creation). We do not intend to mimic Redfish Session expiration, since we feel the added overhead and code complexity is not worth the minimal benefit this detail would provide. Auth tokens are ephemeral in nature, and it is up to the user to recognize this and account for the case of unexpected expiration, whether we implement this detail or not.

The authentication process for users of HTTP Basic Auth will be simple, as this strategy is standards-based (see [RFC7617]). The user will provide base64-encoded credentials with every request to a Redfish endpoint that expects a user to be authorized. Since Ironic supports basic authentication, implementing

this will simply be a matter of passing the users credentials through the pre-existing basic auth middleware. Additionally, if basic auth is in use, the SessionService will be disabled and unusable.

Identity

Since application credentials are scoped upon creation, obtaining the pieces of information that constitute a users identity should be a straightforward process using the existing Ironic policy code and the Keystone middleware. We will use this information to determine what data, actions, etc. the user has access to via the same rules and methods as the existing Ironic API.

It is important to note that with basic auth, such policy-based access restrictions are essentially non-existent. If a user can log in, they will have access to all available data. However, since our basic auth strategy *is* Ironics basic auth, any extension to Ironics basic auth capability would in turn be an extension to the capability of this feature.

Provisioning Tools

The node provisioning tools that will be implemented here shall be functionally identical to existing Bare Metal endpoints, as shown here. The internal logic for achieving this functionality shall mirror that of the actual Ironic API as closely as possible; in theory the only difference should be in how requests by the user and responses to the user are formatted.

Emulated Redfish URI	Equivalent Ironic URI
[GET] /redfish/v1/SystemService/Systems	[GET] /v1/nodes
[GET] /redfish/v1/SystemService/Systems/{uuid}	[GET] /v1/nodes/{uuid}
[POST] /redfish/v1/SystemService/Systems/{uuid} /Actions/ComputerSystem.Reset	**[PUT] /v1/nodes/{uuid}** /states/power

This intermediary will abide by version 1.0.0 of the Redfish spec [RFSHSPEC] and schema [RFSHSCHM] for maximum backwards compatibility with existing tools. More details regarding the planned functionality of these endpoints will be elaborated upon below in the *REST API Impact* section.

Alternatives

The type of BMC interface emulation were looking to implement here does already exist in sushy-tools [SUSHY] and VirtualBMC [VIRTBMC], which emulate Redfish and IPMI respectively. A previous spec was submitted by Tzu-Mainn Chen (tzumainn) which proposed the idea of a sushy-tools driver in Ironic to enable this functionality, but concerns about security, along with the potential value of this existing in Ironic proper have led to the proposal of this spec. [PREVSPEC]

We currently plan on implementing this as a separate WSGI service within the Ironic repository, however it is possible to have both the Ironic API and this Redfish proxy run under the same service. Since both are separate, independent WSGI apps, a WSGI dispatcher, such as the Werkzeug application dispatcher middleware [WSGIDISP] could be used to achieve this.

Data model impact

None.

State Machine Impact

None.

REST API impact

No changes will be made to the Ironic API proper, rather, a new WSGI service hosting a new API will be created as described below. End-users shall be able to interact with this API as if it were a v1.0.0 Redfish endpoint (see [RFSHSPEC] and [RFSHSCHM]).

Since this is a new service, Ironic operators will need to account for the fact that it will need its own port and (if using Keystone) will need to be added as a new endpoint within Keystone. If this proves to be a significant enough inconvenience, however, it could be possible to launch both the Ironic API and this proxy within one service as described above under *Alternatives*.

Redfish API Versions:

- GET /redfish
  - Returns the Redfish protocol version (v1). This will always return the same response shown below, as per the Redfish API spec. (section 6.2 of [RFSHSPEC])
  - Normal response code: 200 OK
  - Example response:

```
{
    "v1": "/redfish/v1/"
}
```

Name	Type	Description
v1	string	The URL of the Redfish v1 ServiceRoot.

- GET /redfish/v1/
  - The Redfish service root URL, will return a Redfish ServiceRoot object containing information about what is available on the Redfish system.
  - Normal response code: 200 OK
  - Example response:

```
{
    "@odata.type": "#ServiceRoot.v1_0_0.ServiceRoot",
    "Id": "IronicProxy",
    "Name": "Ironic Redfish Proxy",
    "RedfishVersion": "1.0.0",
    "Links": {
        "Sessions": {
            "@odata.id": "/redfish/v1/SessionService/Sessions"
```

(continues on next page)

(continued from previous page)

```
        }
    },
    "Systems": {
        "@odata.id": "/redfish/v1/Systems"
    },
    "SessionService": {
        "@odata.id": "/redfish/v1/SessionService"
    },
    "@odata.id": "/redfish/v1/"
}
```

Name	Type	Description
@odata.type	string	The type of the emulated Redfish resource.
@odata.id	string	A resource link.
Id	string	The identifier for this specific resource.
Name	string	The name of this specific ServiceRoot.
Links	object	Contains objects that contain links to relevant resource collections.
Systems	object	Contains a link to a collection of Systems resources.
SessionService	object	Contains a link to the SessionsService resource.
Sessions	object	Contains a link to a collection of Sessions resources.
RedfishVersion	string	The version of this Redfish service.

Sessions

- GET /redfish/v1/SessionService
  - Returns a Redfish SessionService object, containing information about how the SessionService and Session objects are configured.
    - If the underlying Ironic system is using HTTP basic auth, the SessionService will report itself to be disabled, and all Session- related functionality will be non-functional.
  - Normal response code: 200 OK
  - Error response codes: 404 Not Found, 500 Internal Server Error
    - 404 Not Found will be returned if the underlying Ironic system is not using Keystone authentication.
    - 500 Internal Server Error will be returned if the internal request to authenticate could not be fulfilled.
  - Example response:

```
{
    "@odata.type": "#SessionService.v1_0_0.SessionService",
```

(continues on next page)

(continued from previous page)

```
    "Id": "KeystoneAuthProxy",
    "Name": "Redfish Proxy for Keystone Authentication",
    "Status": {
        "State": "Enabled",
        "Health": "OK"
    },
    "ServiceEnabled": true,
    "SessionTimeout": 86400,
    "Sessions": {
        "@odata.id": "/redfish/v1/SessionService/Sessions"
    },
    "@odata.id": "/redfish/v1/SessionService"
}
```

Name	Type	Description
@odata.type	string	The type of the emulated Redfish resource.
@odata.id	string	A resource link.
Id	string	The identifier for this specific resource.
Name	string	The name of this specific resource.
Status	object	An object containing service status info.
State	string	The state of the service, one of either Enabled or Disabled.
Health	string	The health of the service, typically OK.[1]
ServiceEnabled	bool	Indicates whether the SessionService is enabled or not.
SessionTimeout	number	The amount of time, in seconds, before a session expires due to inactivity.[2]
Sessions	object	Contains a link to a collection of Session resources.

- GET /redfish/v1/SessionService/Sessions
  - Returns a Redfish SessionCollection, containing a link to the Session being used to authenticate the request. Requires the user to provide valid authentication in the request header.
  - Normal response code: 200 OK
  - Error response codes: 401 Unauthorized, 404 Not Found, 500 Internal Server Error
    - 401 Unauthorized will be returned if authentication in the header field is either absent or invalid.
    - 404 Not Found will be returned if the underlying Ironic system is not using Keystone authentication.
    - 500 Internal Server Error will be returned if the internal request to authenticate could not be fulfilled.
  - Example response:

[1] This is included for compatibility and should always be OK, although the Redfish schema allows for Warning and Critical as well.

[2] This is a placeholder value. Since sessions are just Keystone auth tokens, they will behave as any other Keystone token, as opposed to behaving like a Redfish Session. (see Authentication)

```
{
    "@odata.type": "#SessionCollection.SessionCollection",
    "Name": "Ironic Proxy Session Collection",
    "Members@odata.count": 1,
    "Members": [
        {
            "@odata.id": "/redfish/v1/SessionService/Sessions/ABC"
        }
    ],
    "@odata.id": "/redfish/v1/SessionService/Sessions"
}
```

Name	Type	Description
@odata.type	string	The type of the emulated Redfish resource.
@odata.id	string	A resource link.
Name	string	The name of this specific resource.
Members@odata.count	number	The number of Session interfaces present in the collection.
Members	array	An array of objects that contain links to individual Session interfaces.

- POST /redfish/v1/SessionService/Sessions
  - Requests Session authentication. A username and password is to be passed in the body, and upon success, the created Session object will be returned. Included in the headers of this response will be the authentication token in the `X-Auth-Token` header, and the link to the Session object in the `Location` header.
  - Normal response code: 201 Created
  - Error response codes: 400 Bad Request, 401 Unauthorized, 404 Not Found, 500 Internal Server Error
    - 400 Bad Request will be returned if the username/password fields are not present in the message body.
    - 401 Unauthorized will be returned if the credentials provided are invalid.
    - 404 Not Found will be returned if the underlying Ironic system is not using Keystone authentication.
    - 500 Internal Server Error will be returned if the internal request to authenticate could not be fulfilled.
  - Example Request:

```
{
    "UserName": "85775665-c110-4b85-8989-e6162170b3ec",
    "Password": "its-a-secret-shhhhh"
}
```

Name	Type	Description
User-Name	string	The UUID of the Keystone application credential to be used for authentication.
Pass-word	string	The secret of said application credential.

– Example Response:

```
Location: /redfish/v1/SessionService/Sessions/identifier
X-Auth-Token: super-duper-secret-aaaaaaaaaaaa

{
    "@odata.id": "/redfish/v1/SessionService/Sessions/identifier",
    "@odata.type": "#Session.1.0.0.Session",
    "Id": "identifier",
    "Name": "user session",
    "UserName": "85775665-c110-4b85-8989-e6162170b3ec"
}
```

Name	Type	Description
@odata.type	string	The type of the emulated Redfish resource.
@odata.id	string	A resource link.
Id	string	The identifier for this specific resource.
Name	string	The name of this specific resource.
UserName	string	The UUID of the application credential used for authentication.

- GET /redfish/v1/SessionService/Sessions/{identifier}
  - Returns the Session with the identifier specified in the URL. Requires the user to provide valid authentication in the request header for the session theyre attempting to access.
  - Normal response code: 200 OK
  - Error response codes: 401 Unauthorized, 403 Forbidden, 404 Not Found, 500 Internal Server Error
    - 401 Unauthorized will be returned if authentication in the header field is either absent or invalid.
    - 403 Forbidden will be returned if authentication in the header field is valid but lacking proper authorization for the Session being accessed.
    - 404 Not Found will be returned if the identifier specified does not correspond to a legitimate Session ID or if the underlying Ironic system is not using Keystone authentication.
    - 500 Internal Server Error will be returned if the internal request to authenticate could not be fulfilled.
  - Example Response:

```
{
    "@odata.id": "/redfish/v1/SessionService/Sessions/identifier",
    "@odata.type": "#Session.1.0.0.Session",
    "Id": "identifier",
    "Name": "user session",
    "UserName": "85775665-c110-4b85-8989-e6162170b3ec"
}
```

Name	Type	Description
@odata.type	string	The type of the emulated Redfish resource.
@odata.id	string	A resource link.
Id	string	The identifier for this specific resource.
Name	string	The name of this specific resource.
UserName	string	The application credential used for authentication

- DELETE /redfish/v1/SessionService/Sessions/{identifier}
  - Ends the session identified in the URL. Requires the user to provide valid authentication in the request header for the session theyre trying to end.
  - Normal response code: 204 No Content
  - Error response codes: 401 Unauthorized, 403 Forbidden, 404 Not Found, 500 Internal Server Error
    - 401 Unauthorized will be returned if authentication in the header field is either absent or invalid.
    - 403 Forbidden will be returned if authentication in the header field is valid but lacking proper authorization for the Session being accessed.
    - 404 Not Found will be returned if the identifier specified does not correspond to a legitimate Session ID or if the underlying Ironic system is not using Keystone authentication.
    - 500 Internal Server Error will be returned if the internal request to authenticate could not be fulfilled.

Node Management

- GET /redfish/v1/Systems
  - Equivalent to `baremetal node list`, will return a collection of Redfish ComputerSystem interfaces that correspond to Ironic nodes. Requires the user to provide valid authentication in the request header for the resource they are trying to access.
  - Normal response code: 200 OK
  - Error response codes: 401 Unauthorized, 403 Forbidden, 500 Internal Server Error
    - 401 Unauthorized will be returned if the authentication in the header field is either absent or invalid.
    - 403 Forbidden will be returned if authentication in the header field is valid but lacking proper privileges for listing Bare Metal nodes.

    * 500 Internal Server Error will be returned if the internal request to the Bare Metal service could not be fulfilled.

  - Example Response:

```
{
    "@odata.type": "#ComputerSystemCollection.
↪ComputerSystemCollection",
    "Name": "Ironic Node Collection",
    "Members@odata.count": 2,
    "Members": [
        {
            "@odata.id": "/redfish/v1/Systems/ABCDEFG"
        },
        {
            "@odata.id": "/redfish/v1/Systems/HIJKLMNOP"
        }
    ],
    "@odata.id": "/redfish/v1/Systems"
}
```

Name	Type	Description
@odata.type	string	The type of the emulated Redfish resource.
@odata.id	string	A resource link.
Name	string	The name of this specific resource.
Members@odata.count	number	The number of System interfaces present in the collection.
Members	array	An array of objects that contain links to individual System interfaces.

- GET /redfish/v1/Systems/{node_ident}
  - Equivalent to `baremetal node show`, albeit with fewer details. Will return a Redfish System resource containing basic info, power info, and the location of the power control interface. Requires the user to provide valid authentication for the resource they are trying to access.
  - Normal response code: 200 OK
  - Error response codes: 401 Unauthorized, 403 Forbidden, 404 Not Found, 500 Internal Server Error
    * 401 Unauthorized will be returned if the authentication in the header field is either absent or invalid.
    * 403 Forbidden will be returned if authentication in the header field is valid but lacking proper privileges for the Bare Metal node being accessed.
    * 404 Not Found will be returned if the identifier specified does not correspond to a legitimate node UUID.
    * 500 Internal Server Error will be returned if the internal request to the Bare Metal service could not be fulfilled.
  - Example Response:

```
{
    "@odata.type": "#ComputerSystem.v1.0.0.ComputerSystem",
    "Id": "ABCDEFG",
    "Name": "Baremetal Host ABC",
    "Description": "It's a computer",
    "UUID": "ABCDEFG",
    "PowerState": "On",
    "Actions": {
        "#ComputerSystem.Reset": {
            "target": "/redfish/v1/Systems/ABCDEFG/Actions/
↪ComputerSystem.Reset",
            "ResetType@Redfish.AllowableValues": [
                "On",
                "ForceOn",
                "ForceOff",
                "ForceRestart",
                "GracefulRestart",
                "GracefulShutdown"
            ]
        }
    },
    "@odata.id": "/redfish/v1/Systems/ABCDEFG"
}
```

Name	Type	Description
@odata.type	string	The type of the emulated Redfish resource.
@odata.id	string	A resource link.
Id	string	The identifier for this specific resource. Equal to the corresponding Ironic node UUID.
Name	string	The name of this specific resource. Equal to the name of the corresponding Ironic node if set, otherwise equal to the node UUID.
Description	string	If the Ironic node has a description set, it will be returned here. If not, this field will not be returned.
UUID	string	The UUID of this resource.
PowerState	string	The current state of the node/System in question, one of either On, Off, Powering On, or Powering Off.
Actions	object	Contains the defined actions that can be executed on this system.
#ComputerSystem. Reset	object	Contains information about the Reset action.
target	string	The URI of the Reset action interface.
ResetType@Redfish. AllowableValues	array	An array of strings containing all the valid options this action provides.

- POST /redfish/v1/Systems/{node_ident}/Actions/ComputerSystem.Reset
  - Invokes a Reset action to change the power state of the node/System. The type of Reset

action to take should be specified in the request body. Requires the user to provide valid authentication in the request header for the resource they are attempting to access.

- Accepts the following values for ResetType in the body[3]:
  - On (soft power on)
  - ForceOn (hard power on)
  - GracefulShutdown (soft power off)
  - ForceOff (hard power off)
  - GracefulRestart (soft reboot)
  - ForceRestart (hard reboot)
- Normal response code: 202 Accepted
- Error response codes: 400 Bad Request, 401 Unauthorized, 403 Forbidden, 404 Not Found, 409 NodeLocked/ClientError, 500 Internal Server Error, 503 NoFreeConductorWorkers (for more on codes 409 and 503, see the details for PUT requests to `/v1/nodes/{ident}/states/power` in [IRONCAPI]):
  - 400 Bad Request will be returned if the ResetType field is not found in the message body, or if the field has an invalid value.
  - 401 Unauthorized will be returned if the authentication in the header field is either absent or invalid.
  - 403 Forbidden will be returned if authentication in the header field is valid but lacking proper privileges to perform the specified action on the Bare Metal node being accessed.
  - 404 Not Found will be returned if the identifier specified does not correspond to a legitimate node UUID.
  - 409 NodeLocked/ClientError is an error code specified in the Bare Metal API call this request is proxied to. The body of a 409 response will be the same as that which was received from the Bare Metal API.
  - 500 Internal Server Error will be returned if the internal request to the Bare Metal service could not be fulfilled.
  - 503 NoFreeConductorWorkers is an error code specified in the Bare Metal API call this request is proxied to. The body of a 503 response will be the same as that which was received from the Bare Metal API.
- Example Request:

```
X-Auth-Token: super-duper-secret-aaaaaaaaaaaa

{
    "ResetType": "ForceOff"
}
```

[3] The Redfish schema for ResetType also includes Nmi (diagnostic interrupt) and PushPowerButton (simulates a physical power button press event). However, neither of these actions are part of Ironics node power state workflow and support for these actions varies greatly depending on the driver and hardware.

Name	Type	Description
ResetType	string	The type of Reset action to take (see above)

Client (CLI) impact

None.

openstack baremetal CLI

Though this addition would include new REST API endpoints, this feature merely provides another way for users to access already existing features within the Ironic API, which are already accessible from the `openstack baremetal` CLI.

openstacksdk

None.

RPC API impact

None.

Driver API impact

None.

Nova driver impact

None.

Ramdisk impact

None.

Security impact

The main consideration when it comes to the security of this feature is the addition of a new means of accessing Ironic hardware. However, this should not pose much of a new concern when it comes to security, since authentication shall be implemented in the same way and using the same midddlewares as the existing Ironic API. Nevertheless, great care will be taken to make sure that said integration with the existing auth middleware is safe and secure.

To mitigate further risk, generated application credentials can and should be limited in scope to only allow access to the parts of the Ironic API required by this intermediary. We will also require all requests to be performed over HTTPS, since session tokens, application credential secrets, and (base64- encoded) HTTP basic auth credentials will be sent in plain text.

Other end user impact

This will give end users an alternative way of accessing power controls, one compatible with existing Redfish provisioning tools. This means in theory, the majority of users wont be making API calls directly, instead utilizing pre-existing Redfish-compatible software, such as Redfishtool.

Scalability impact

None.

Performance Impact

Since this shall be implemented as a separate WSGI service, some additional overhead will be required, although the impact should be minor, as it will not require the running of any periodic tasks nor the execution of any extraneous database queries.

Additionally, it should be noted that running this proxy service is completely optional on the part of the Ironic system operator; if one does not wish to use it, its existence can simply be ignored.

Other deployer impact

This feature should have no impact on those who do not wish to use it, as it must be ran separately and can be ignored. To prevent it from being started accidentally, operators shall be able to explicitly disable it in the Ironic configuration file.

Those who wish to make use of this feature must keep in mind that since it is currently being implemented as a separate WSGI service, it shall require at minimum its own port to be ran on. This can be useful if one wishes to have the Redfish proxy service bound to a different port or a different host IP from the Ironic API; however, it will require a new endpoint to be added via Keystone (if using Keystone), and may potentially require extra network configuration on the part of the system administrator.

Developer impact

This new service shall be implemented in Flask, as opposed to Pecan, which is what the Ironic API currently uses. As such, all code written for this new feature shall be well-documented in order to maximize its readability to any Ironic devs unfamiliar with Flask.

It has been mentioned by TheJulia that in future, the Ironic dev team may want to look into migrating the Ironic API to use Flask, and this addition to the codebase may prove useful to those tasked with said migration.

Finally, the Sessions feature does not exist in sushy-tools; since this spec includes a planned implementation of it, it could possibly be a useful addition there. This basic Redfish proxy can also be extended in future to provide access to even more parts of an Ironic system through a Redfish-like interface to those who would find such functionality useful.

3.9.3 Implementation

Assignee(s)

Primary assignee:

Sam Zuk (sam_z / szuk) <szuk@redhat.com>

Other contributors:

Tzu-Mainn Chen (tzumainn) <tzumainn@redhat.com>

Work Items

- Create the necessary API endpoints
  - Implement the Redfish System -> Ironic Node proxy

  - Implement the Redfish Session -> Keystone authentication proxy
  - Write unit tests and functional tests to ensure proper functionality
- Write documentation for how to use and configure this functionality, for users, administrators, and developers.
- Test this feature on real hardware in a way that mimics expected use cases.

3.9.4 Dependencies

None.

3.9.5 Testing

Functional testing will be required to ensure requests made to these new proxy endpoints result in the correct behavior when ran on an actual Ironic setup. Furthermore, rigorous test cases should be written to make extremely sure that no unauthorized access to node APIs is possible.

3.9.6 Upgrades and Backwards Compatibility

N/A

3.9.7 Documentation Impact

Documentation will need to be provided for the new API endpoints, along with the necessary instructions for how to enable and configure this feature (for operators), along with additional information end users may require, such as how to work with authentication tokens.

3.9.8 References

3.10 Kea DHCP backend

https://bugs.launchpad.net/ironic/+bug/2081847

3.10.1 Problem description

DHCP is a critical service for assigning IP addresses to instances in OpenStack deployments. Both Ironic and Neutron rely on dnsmasq to provide DHCP services. However, dnsmasq has become increasingly unreliable, with intermittent DHCP failures, frequent restarts and crashes of dnsmasq processes, etc.

Refer to the bug on dnsmasq here: https://bugs.launchpad.net/ironic/+bug/2026757.

3.10.2 Proposed change

Add a new Kea DHCP backend for Ironic and Neutron as an alternative DHCP provider. This will provide a more reliable DHCP option thats easily extensible while maintaining compatibility with existing systems.

- Create a new DHCP provider class extending the BaseDHCP interface.
- Add configuration options to enable and configure Kea DHCP.
- Add methods to manage DHCP options, leases, and port updates using Kea APIs.
- Ensure DevStack support for Kea
- Update documentation to cover the addition of Kea DHCP, setup and usage.

Ironic will interact with the Kea DHCP server via HTTP-based APIs[1].

- `config-get`, `config-set`: Manage Kea configuration
- `reservation-get`: Retrieve host reservations that will be handled externally via a host-reservation database shared between Kea and Ironic/Neutron query to retrieve reservation data. Dynamic reservation updates (`reservation-add` and `reservation-del`) require a premium ISC subscription and are not supported in this implementation[3].
- `lease4-get`, `lease6-get`: Retrieve lease information
- `subnet4-add`, `subnet6-add`: Add new subnets
- `statistic-get`: Monitor DHCP server statistics

The Kea DHCP backend can be added with or without an agent layer. Both approaches will use Keas HTTP-based APIs, but differ in their architecture and how Ironic interacts with the Kea DHCP server.

An agent approach has more moving parts and complexity, but better scalability for large deployments, local management of Kea instances and reduced network traffic to central Ironic service.

- Flow: Ironic Driver Agents Kea Servers
- Kea DHCP Agent is deployed on each DHCP server node and manages local Kea config and handles communication between Ironic and Kea.
- Ironic DHCP Driver implements Ironic DHCP interface and communicates with the Kea DHCP Agent.
- Ironic sends DHCP configuration to Ironic DHCP Driver which in turn distributes it to Kea agents to apply config to local Kea servers.
- For lease management, agents monitor local Kea lease changes report changes back to Ironic DHCP Driver.

The agentless approach will have a much simpler architecture, probably easier to implement and maintain with direct control from Ironic, but may not scale as well, increased network traffic to Kea servers and even a potential performance impact on Ironic for large-scale operations.

- **Flow: Ironic driver translates configs to Kea API calls and applies changes** directly to Kea servers.
- Ironic manages all Kea servers, through direct Kea API no agent layer.
- The Ironic driver translates configs to Kea API calls and applies changes directly to Kea servers.
- For lease management, maybe a periodic polling of Kea servers for lease updates, or, a webhook mechanism.

Alternatives

- Find a way to improve existing Dnsmasq implementations. Its worth noting that dnsmasq has inherent limitations as its really intended for small computer networks even though Ironic has been using it for the longest, so addressing these problems may require significant re-engineering. Its also single maintainer OSS project, as of the time of this specification. The fact it is a single maintainer project results in long spans of inactivity, which increases consumer risk for a project like OpenStack.

[1] https://kea.readthedocs.io/en/latest/api.html
[3] https://kea.readthedocs.io/en/latest/arm/hooks.html

- Use a different DHCP provider than Kea that is also actively maintained, provides the reliability, flexibility, and integration Kea offers or better.
- Develop a completely new, custom DHCP server which will be quite an undertaking and additional long-term technical debt.

Data model impact

No changes to Ironics data model are required.

State Machine Impact

No changes to the state machine are required.

REST API impact

No changes to the REST API are required.

Client (CLI) impact

No changes to python-ironicclient are required.

RPC API impact

No changes to the RPC API are required.

Driver API impact

A new DHCP provider class will be added, but this should not impact existing drivers.

Nova driver impact

None

Ramdisk impact

No changes to the ironic-python-agent or ramdisk are required.

Security impact

Theres no expected security trade-offs with the addition of a Kea DHCP backend. The increase in options for operators limits overall risk by providing additional options, which should be a net security gain.

In the event of such occurrences in the future, its active development will likely ensure timely security updates.

Other end user impact

None

Scalability impact

Kea DHCP is designed for better scalability than dnsmasq, which could improve performance, especially for large deployments.

Performance Impact

Kea DHCP will likely offer performance improvements over dnsmasq, especially for large deployments with thousands of machines.

Other deployer impact

Deployers will need to install and configure the Kea DHCP server alongside Ironic, likely in the same manner as dnsmasq, but with Kea-specific configurables such as network interfaces, IP address ranges, and lease times.

Developer impact

None

3.10.3 Implementation

Assignee(s)

Primary assignee:
Afonne-CID (cid).

Other contributors:
Jay Faulkner (JayF).

Work Items

- Write a Kea DHCP backend extending the BaseDHCP interface.
- Add unit tests and DevStack support for running with Kea.
- Configure at least one CI job to use Kea DHCP.
- Add Bifrost support for the new Kea DHCP backend.
- Implement unmanaged inspection support for Kea DHCP.
- Update documentation.

3.10.4 Dependencies

- Kea DHCP server[2].

3.10.5 Testing

Add tests to verify full DevStack support, new Tempest tests specific to Kea DHCP functionality, and new unit tests for: * Retrieving lease information via Kea APIs. * Ensuring Kea correctly reads from the host-reservation database.

3.10.6 Upgrades and Backwards Compatibility

Existing dnsmasq support will remain unchanged.

[2] https://www.isc.org/kea/

3.10.7 Documentation Impact

- Full documentation on Keas capabilities and how its different from dnsmasq with cross-references to external Kea resources.
- Document configuration options and steps on switching from dnsmasq to Kea, including configuring Kea to use a read-only host-reservation database and how to set up Ironic/Neutron to manage this database.
- Installation, configuration, and architecture documentations should present Kea as a configurable option, with clear instructions on how users can choose between Kea and dnsmasq.
- API documentation will need to be updated to reflect any changes to existing methods and how they now interact with Kea compared to dnsmasq.
- Sections of the current documentation that might have referenced dnsmasq as the default or only DHCP provider will need to be updated to reflect that Kea is now also a supported backend.

3.10.8 References

3.11 Project Mercury

https://bugs.launchpad.net/ironic/+bug/2063169

This is a project to create an simplified framework between Ironic and physical network configuration to facilitate orchustration of networking in a delineated way from existing OpenStack Neutron service in a model which would able to operated effectively by another team which is not a cloud team, but a network team.

The reasons why are plentiful:

- The number of Operators utilizing Ironic continue to grow, although the operators utilizing Ironic in fully integrated configurations is not growing at the same rate as operators running in a standalone mode.
- Operators needing physical switch management generally need to operate in an environment with strong enforcement of separation of duties. i.e. The software might not be granted access to the Switch management framework, nor can such a service be accessible by any users under any circumstances.
- The introduction of DPUs generally means that we now have potential cases where switches need to be programmed to provision a DPU, and then the DPU needs to be programmed to provision servers.

The goals can be summarized as:

- Provide a mechanism to configure L2 networks on Switches, which may be facilitated by a modified networking-generic-switch or similar plugin.
- Provide a mechanism to configure L2 networks to be provided to a host from a DPU.
- Provide a mechanism to accommodate highly isolated network management interfaces where operators restrict access such that *only* Ironic is able to connect to the remote endpoint.
- Provide a tool to apply and clean-up configuration, *not* track and then assert configuration. This doesnt preclude a future double check this configuration mode from existing at some point, but the minimal viable functionality is application and removal of the network configuration.

- Provide a mechanism of receiving the call to do something, reading in networking configuration credentials from local storage, and then doing so without the need of a database *OR* shared message bus.

This project is NOT:

- Intended to provide any sort of IPAM functionality.
- Intended to provide management of Routing.
- Intended to provide management of Security Groups or Firewalling.
- Intended to provide a public ReSTful API, nor require a database.

This project MAY:

- Provide a means to help enable and deploy advanced tooling to a DPU under Ironics control.
- Provide a means of offloading some of the layer2 interaction responsibility in an environment *with* Neutron and Ironic, espescially.

Warning

This document is not a precise and thus prescriptive design document, but an document to record and surface the ideas in a way that can foster communication and consensus building. In this case, we are likely to leave it to the implementers progotive with this document being overall guard rails.

3.11.1 Problem description

Today, Ironic has only one option to facilitate the automation of switch level infrastructure, which is to leverage Neutron and the ML2 interface. Unfortunately, infrastructure operators needing Ironic largely reject this model because the network is often owned by a separate group.

As a result we need a service to facilitate secure and delienated network management which can be owned and operated by separate infrastructure team in an enterprise organization, which brings together, aspects like simple code patterns and playbooks such that they can trust the interface layer to apply basic network configuration and enable easier use of Bare Metal. In most cases where this is needed, just the physical port needs to be set to a specific network; addressing is often separately managed and not our concern.

Furthermore, the available ecosystem in the DPU spaces wants to model their devices in a variery of ways and some of those devices have inherent limitations. For example, some devices are just another computer embedded and connected to the same PCI device bus. Others present the ability to load a P4 program to handle specific tasks. Or the available Flash and RAM of a device is highly limited such that options are very limited and entirely exclude all off the shelf operating systems and their utilities. This nature makes almost every device and their resulting use case entirely govern their configuration and use model which means an easy to modify modular interface would provide the greatest potential impact.

3.11.2 Proposed change

We are proposing an RPC service. Specifically something along the lines of a JSON-RPC endpoint, which multiple ironic conductors would be able to connect to in order to request networking changes to be made.

Along side of the RPC service, we would have an appropriately named `network_interface` driver to take the information stored inside of Ironic and perform attachment of interfaces based upon the provided information.

Note

A distinct possibility exists that we may actually start with a hybird dual `network_interface` driver to help us delineate and handle integration in a clear fashion. This is much more of an implementors progorative soft of item.

MVP would likely exclude locking, but be modeled as a single worker service or container which does not maintain state, largely simplifies the problem to who is logged into what to make concurrent changes, which has been the historical driver for locking.

Note

Teaching networking-baremetal to call this proposed RPC service is generally considered out of scope of this work, but entirely within reason and possibility to facilitate as this would provide functionally a capability for some calls to be proxied through and related to actions for a Neutron deployment to ultimately also call this new service.

The overall call model, at a high level could take the following flow.:

```
Inbound Request/Connection



{"type": "attach_port",
 "payload": {"context": {...}}}



Invoke Plugin With Context



Plugin handles locking, if necessary



Plugin succeeds (HTTP 200?) or fails and returns HTTP error
```

While originally envisioned to just be able to load an ML2 plugin directly, there plugins design model has some challenges in this context of remote execution, which is likely why a remote RPC model never evolved in Neutron.

Two basic issues:

1) Plugins, upon completing the binding action, update the database state in neutron through a pattern of updating the neutron database. This requires database access and credentials which are not available in our use case and model.

2) Plugins may also invoke methods on the original provided context. Context in Neutron, in this

case, is not context provided by oslo.context, but an entirely separate construct consisting of past and future state information for the networking being modified.

So the obvious path forward is to simplify the model and design the RPC model used for this interaction around the basic actions, and allow for an ML2 user, the ability for the calls to be made which abstract the actions from the information/state/configuration updates.

- get_allowed_network_types - Returns a list of supported network types which can allow the caller to determine if the action can be supplied. Presently, baremetal engaging ML2 plugins largely just hard-code this as only supporting a VLAN type, but pushing this as far out as possible to a plugin being executed allows for it to be a generic pass-through.
- is_link_valid - Returns True/False depending on if the request has sufficient and correct information to be acted upon. Could be used for basic pre-action validation.
- is_port_supported - Returns True/False depending if the requested port is supported for actions. Generally this is a VNIC type check today, but could also be used by the remote service to perform basic pre-fliflight validation actions and allow a client to fail-fast.
- attach_port - Performs the actual action of attaching (Adding vlans or ultimately VNIs to a port).
- detach_port - Performs the actual action of detaching (removing network access from a port).
- update_port - Peroforms an attempt to update a port for attachment, such as if port-channels/bonding properties have changed.
- add_network - Adds a network to the remote device.
- delete_network - Deletes the network from the remote device.

In a sense, this changes the idea of just load an ml2 plugin to load either a hybridized ml2 plugin interface, OR just define our own plugin model which can be supported, and as such is an outstanding question this specification seeks to bring an answer to.

For the remote RPC service, it is anticipated that logging will need to be verbose enough that Operators can understand the questions they may raise when investigating issues. For example: When, Who, What, Why, and How. Plugin code in Ironic should **also** log verbosely when invoked to ensure operators can match requests and resulting changes should an issue arise.

While beyond an initial MVP of basic functionality, to solve the DPU case, the overall pattern model would likely take the shape of one where Ironic would enumerate through the child nodes, attach the child nodes to the requested physical network, and then engage on some level of programming which may need to be vendor or deployment specific based upon the overall use model. Details which at present time cannot be determined without the foundational layer needing to be constructed before being built upon.

From a users standpoint, the following sequence depicts their basic interaction and the overall resulting sequence.:

```
Existing node chosen by user



User posts to /v1/nodes/<ident>/vifs with payload
containing an id of a vlan ID.
```

(continues on next page)

(continued from previous page)

```
User requests the node to deploy via the
/v1/nodes/<ident>/states/provision API interface



Ironic follows existing flow, triggering the new
network_interface module which calls this new
service to perform the attach/detach operations
in accordance with the existing model and node
lifecycle state.
Initial network in a deploy is the provisioning
network.



Node deployment proceeds with resources already
connected to the desired network.



Once deployment has been completed, the network
interface module calls the new service to change
the attachment to the requested vlan ID. In the
event of a failure, the physical switch port
is detached.
```

Alternatives

The closest alternative would be a `standalone Neutron` coupled with some sort of extended/proxy RPC model, which is fine, but that really does not address the underlying challenge of the attach/detach functionality being needed by Infrastructure Operators. It also introduces modeling which might not be suitable for bulk infrastructure operators as they would need to think and operate a cloud model, as opposed to the physical infrasructure model. Plus operating Neutron would require a database to be managed, increasing operational complexity, and state would also need to still be navigated which increases the configuration and code complexity based upon different Neutron use models.

Another possibility would be to directly embed the network attach/detach loading and logic into Ironic itself, however that would present difficulties with maintenance where we largely want to unlock capability.

Data model impact

At this time, we are largely modeling the idea to leverage existing port data stored inside of Ironic which is utilized for attachment operations.

A distinct possibility exists we may look at storing some additional physical and logical networking detail inside of Ironics database to be included in requests, which could possibly be synchronized, but this would be beyond the scope of the minimum viable product as in the initial phase we intend to use

the VIF attach/detach model to represent the logical network to be attached.

State Machine Impact

None

REST API impact

With a MVP, we do not anticipate any REST API changes to Ironic itself with the very minor exception of the losing of a Regular Expression around what Ironic accepts for VIF attachment requests. This was agreed upon by the Ironic community quite some time ago and just never performed.

Existing fields on a node and port will continue to be used just as they have before with an MVP.

Post-MVP may include some sort of /v1/physical_network endpoint to be designed, but that is anticipated to be designed once we know more.

Client (CLI) impact

openstack baremetal CLI

None

openstacksdk

None

RPC API impact

This change proposes a service which would be accessed by Ironic utilizing an RPC model of interaction. This means there would be some shared meaning for call interactions in the form of a library.

In all likelihood, this may be as simple as attach and detach, but given the overall needs of an MVP and a use model were focusing upon trying to leverage existing tooling as well, the exact details are best discovered through the development process which likely covers what was noted above in the Proposed Change section.

Driver API impact

None

Nova driver impact

None

Ramdisk impact

None

Security impact

Impact for Ironic itself is minimal, although it will require credentials to be set for the remote service to signal interface attachment/detachments.

The security risk largely revolves around the new service were looking at creating with this effort. The shared library utilized to connect to the remote service, likely needs to also contain the necessary client

wrapper code, as an MVP service is likely going to start only with support for HTTP Digest Authentication, which can then move towards certificate authentication as it evolves.

In large part because that service will need to load and combine a set of credentials and access information. As such, this new service will **not** be a user facing service.

Today, individual ports are attached through a combination of network identifier and a binding profile which is utilized to map a port to a switch. In this model, there would be no substantial difference. A network_id would be a user supplied detail, and the local_link_information would contain sufficient information for the plugin executing to identify which device. The new service would retrieve the details to access the remote device from local configuration, and combine the rest of the binding profile and target network identifier to facilitate the attachment of the port to the device.

Note

This security impact does not denote the likely situation of DPU credential management. We are presently defferring the possibility as a challenge we would focus on after an initial minimum viable product state is reached.

Note

This security risk does not include any future mechanisms to do perform aspects such as software deployment on a DPU to facilitate a fully integrated with Neutron case, which is something we would want to identify and determine as we iterate along the path to support such a capability.

Other end user impact

None

Scalability impact

Please see the Performance Impact section below.

Performance Impact

This proposal is intentionally designed to be limited and isolated to minimize risk and reduce deployment complexity. It is also intentionally modeled as a tool to do something, and that something happens to be configuration in area where device locking is necessary. This realistically means that the only content written to disk is going to be lock files.

Furthermore, the possibility exists that the Ironic driver code utilized *could* wait for a response, where today Neutron port attachment/detachment calls are asynchronous. This would pose an overall improvement for end users of Ironic. This is solved today as a 15 second sleep by default, and might not be necessary in this design overall improving Ironic performance.

Other deployer impact

To utilize this functionality, deployers would need to deploy a new service.

This would be opt-in, and would not impact existing users.

Developer impact

None

3.11.3 Implementation

Assignee(s)

Primary assignee:
<Volunteer #1>

Other contributors:
<Volunteer #2>

Work Items

Broadly speaking, the work items would include:

1) Prototyping this new service.
2) Wire up an ML2 driver such that we have interfaces we can load and call. This is anticipated to be neworking-generic-switch.
3) Prototyping an ironic network_interface driver to utilize this new service.
4) Test!

Note

The list below is intended to paint a picture of what we feel are the possible steps beyond the initial step of creating an Minimum Viable Product. They are included to provide a complete contextual picture to help the reader understand our mental model.

Past initial prototyping, the following may apply order:

- Creation of a common library for Ironic and any other program or tool to utilize to compose RPC calls to this service.
- Extend support to VXLAN ports, which may require additional details or design work to take place and work in any ML2 driver utilized.
- Design an API rest endpoint to facilitate the tracking of physical networks to be attached to baremetal nodes.
- Add support to networking-baremetal to try and reconcile these physical networks into Ironic, so node port attachment/detachments can take place.
- Add support to networking-baremetal for it to proxy the request through to this service for port binding requests in Neutron.
- Design a new model, likely superseding VIFS, but vifs could just also be an internal network ID moving forward. This would likely be required for formal adoption of the functionality by Metal3, but standalone users could move to leverage this immediately once implemented.
- Development of a model and flow where DPU devices could have a service deployed to them as part of a step invoked by Ironic. This would involve many challenges, but could be used to support the Neutron Integrated OVS/OVN agents to operate on the card for cases such as the remote card being in a hypervisor node.

3.11.4 Dependencies

To be determined.

3.11.5 Testing

An ideal model of testing in upstream CI has not been determined, and is dependent upon the state upon reaching a minimum viable product state, and then what the next objectives appear to be.

This may involve duplication of Ironics existing multinode job in a standalone form. Ultimately the expectation is we would have one or more CI jobs dedicated to supporting such functionality being exercised.

3.11.6 Upgrades and Backwards Compatibility

This functionality is anticipated to be net new for Ironic and exposed to end users through a dedicated `network_interface` module which could be selected by users at a point in the future. As such no upgrade or backwards compatibility issues are anticipated.

3.11.7 Documentation Impact

No impact is anticipated at this time.

3.11.8 References

https://etherpad.opendev.org/p/ironic-ptg-april-2024#L609

3.12 Merge Inspector into Ironic

https://storyboard.openstack.org/#!/story/2010275

This specification proposes merging the ironic-inspector project (*Inspector*) into the main Ironic project completely, making all its API feature a part of the Bare Metal API and deprecating the separate project.

3.12.1 Problem description

Inspector was born from a bunch of unmerged patches to Ironic back in the middle of year 2014 under the name *ironic-discoverd*. The Ironic team considered the patches, especially the idea of auto-discovery, going too much outside of the Bare Metal project scope. In early 2015, the Ironic team has agreed to move the already established and functioning project under its wing, changing the name to *ironic-inspector*. Over time, Inspector gained and then lost again its own core reviewer team, while becoming more and more aligned with the standard practices in the Ironic community.

Fast forward to year 2023, very few still standing members of the community remember this story. Why is Inspector separate from Ironic? is a common question among newcomers. There are few problems with the current state of things:

Maintenance load.
: A great example is the SQLAlchemy 2.0 transition. Since *Inspector* has a separate database, we need to do the work twice.

Operating load.
: One more service - one more problem. It has to be installed, secured and updated regularly.

Scaling/HA issues.
: Ironic has been designed with horizontal scaling and high availability in mind; Inspector has not.

To a certain extent, it makes sense: Inspector is only used occasionally. Still, in a really large deployment (like CERNs) even rarely used services are accessed often enough for the availability to matter.

The problem has been mostly mitigated by introducing support for a message bus (RabbitMQ) and membership tracking (ZooKeeper/etcd). Mostly - because the solution depends on two services that are normally present in OpenStack, but are not present in standalone solutions like Bifrost or Metal3[1].

Performance issues.
Since Inspector has a separate database, it has to maintain its own nodes table and synchronize it with Ironic periodically. This is inefficient and may contribute to scalability problems.

Similarly, every inspection hook (see the *Glossary*) has to issue some API requests to Ironic to update information. A pipeline with several hooks may generate noticeable traffic.

Resource utilization.
Inspector, being written in Python, has a non-negligible RAM footprint, while doing nearly nothing most of the time.

Discoverability.
Since Inspector has its own documentation, API reference and client, its harder to discover information about it.

Glossary

In-band inspection
Hardware inspection that is conducted through the use of a ramdisk with IPA (ironic-python-agent). The other option is out-of-band inspection, where the data is collected through the BMC without powering on the node.

The term *introspection* is used as a synonym in the Inspector code base and documentation.

Inspection data
Generic term for all information collected by IPA during inspection and sent back to Inspector (in the future - to Ironic).

Inventory
In the broad sense - synonym to inspection data. In the narrower sense - hardware inventory as defined by IPA and returned by its hardware managers.

Inspection collectors
IPA plugin responsible for collecting inspection data on the ramdisk side. The `default` one collects inventory. Collectors are currently independent from hardware managers, but this may change in the future.

Processing hooks
Inspector (in the future - Ironic) plugins for processing inspection data and updating the node as a result. Allow operators to fine-tune the inspection process by configuring a pipeline of processing hooks.

This document introduces the term *inspection* hooks to avoid ambiguity in the larger Ironic context.

Inspection hooks and collectors may correspond to each other. For example, the `extra-hardware` collector has a corresponding `extra_hardware` hook.

[1] Technically, any Kubernetes deployment includes etcd, but it is against best practices for non-Kubernetes applications to rely on it.

Managed/unmanaged inspection
We are talking about *managed* inspection when it is Ironic that completely sets up the boot environment, be it (i)PXE or virtual media boot. Inspection is *unmanaged* when the (i)PXE environment from Inspector, usually a separate *dnsmasq* installation, is used.

Unmanaged inspection has been the default for Inspector for a long time. Managed inspection was introduced to support virtual media, first and foremost. With the proposed merge, managed inspection will become the default mode, while unmanaged inspection will require an explicit configuration.

PXE filters
A way for the separate Inspectors PXE environment to co-exist with Neutron by limiting which MAC addresses it serves. Inspector has two implementations:

- `iptables` uses firewall to open/close access to a DHCP server
- `dnsmasq` updated *dnsmasq* host files with MAC addresses to serve/deny

Over time, the `dnsmasq` PXE filter was found to be more scalable and flexible (e.g. it supports spine/leaf architectures with DHCP relays) at the cost of supporting only one DHCP server implementation.

This specification proposes migrating only the `dnsmasq` PXE filter. The `iptables` one will follow if we see a demand for it (the deprecation release notes will mention it).

Auto-discovery
Process of enrolling new nodes automatically. When unmanaged inspection is configured, unknown nodes on the provisioning network will boot into the IPA ramdisk, get inspected and registered in Ironic.

This operation is disabled by default.

3.12.2 Proposed change

Inspector features will be migrated step-by-step into the Ironic code base, predominantly into the `ironic.drivers.modules.inspector` package. We will try to avoid radical changes in the design with the following exceptions:

- Split the poorly defined *inspection data* into the formally defined and version (although not in the sense of API microversions) inventory and free-form plugin data (influenced by inspection collectors in IPA and processing hooks in Inspector/Ironic - see the *glossary*).

  Inventory will not be mutated by any inspection hooks, thus reducing the need for having unprocessed inspection data API (which Inspector has).

- Avoid poorly documented internals data formats in the processed data. For example, Inspector generates fields `interfaces` and `all_interfaces` that are not based on the `interfaces` collection in the inventory.

- Split the migrated PXE filter into a new script to avoid coupling it (and thus the *dnsmasq* instance behind it) to Ironic processes. This way, it can be scaled separately.

- Rework inspection (former processing) hooks for more obvious naming and better composability.

  Consistently use dashes instead of underscores in entry point names.

  Fewer hooks will run in the default configuration.

- Consistently use the term *inspection* instead of *introspection*.

For the sake of keeping this specifications size reasonable and my sanity (relatively) intact, inspection rules are omitted here. Theyre relatively trivial, but require a lot of explanation and can be implemented independently.

Alternatives

Keep Inspector separate.
Possible arguments for it include:

- Better utilizing CPU cores by having a separate process. This should better be solved by allowing several conductors per physical host. Such a solution will benefit also more intensive operations and deployments without Inspector.
- More manageable (i.e. smaller) code base. However, a lot of code in Inspector exists only because its a separate project. This includes most of the database code, some of the API endpoints and the node synchronization routine.

Do not migrate some of the major features.
This will hinder the migration and will prevent us from ever deprecating Inspector.

Do not migrate PXE filters.
This will make auto-discovery in the presence of Neutron impossible. Auto-discovery is a commonly requested feature (example request).

Data model impact

Inventory table

Add a new table for storing inspection data when the Object Storage service is not available:

```
class NodeInventory(Base):
    """Represents an inventory of a baremetal node."""
    __tablename__ = 'node_inventory'
    __table_args__ = (
        Index('inventory_node_id_idx', 'node_id'),
        table_args())
    id = Column(Integer, primary_key=True)
    inventory_data = Column(db_types.JsonEncodedDict(mysql_as_long=True))
    plugin_data = Column(db_types.JsonEncodedDict(mysql_as_long=True))
    node_id = Column(Integer, ForeignKey('nodes.id'), nullable=True)
```

Here, `inventory_data` contains the inventory as defined by IPA (see the *Glossary*), while `plugin_data` contains auxiliary data returned by various collectors or generated by inspection hooks.

The `NodeInventory` object is deleted on node deletion.

> **Note**
>
> This table already exists at the time of writing this specification. It is included here for completeness.

Node modifications

Add a new boolean field `auto_discovered` to the `nodes` table. It will be read-only from the API standpoint and will be used to mark auto-discovered nodes.

State Machine Impact

The only expected change is a possibility of transition from *INSPECT WAIT* to *INSPECTING*, which is arguably missing by mistake.

REST API impact

Add a new API endpoint to fetch the inventory and the optional plugin data:

`GET /v1/nodes/{node}/inventory`
Returns a JSON object with two keys: `inventory` and `plugin_data`.

HTTP status codes:

- 200 on success.
- 404 if the node is not found, no inventory is recorded, or the API is not available in the requested version.

> **Note**
>
> This API already exists at the time of writing this specification. It is included here for completeness.

Add a new API to accept the inspection data from the ramdisk:

`POST /v1/continue_inspection`
Accepts inspection data in exactly the same format as Inspector, namely as a JSON object with at least an `inventory` field.

The only query parameter is `node_uuid` - an optional UUID of the node. This parameter is designed for virtual media deployments to safely pass the node identity to the ramdisk. See *lookup process* for some details.

This API is **not authenticated** and does not require an agent token since inspection always happens first on agent start-up.

The result depends on the requested API version:

- In the base API version, Inspector compatibility mode is used. The resulting JSON object has one key: `uuid` (the nodes UUID).
- In the new API version, the result is the same as in the normal lookup API. In this case, the API generates and returns an agent token, effectively replacing lookup when inspection is used.

HTTP status codes:

- 200 on success.
- 404 if the node is not found, several nodes match the provided data, or the node state is not `INSPECT WAIT`.

> **Note**
>
> We use the same generic HTTP 404 response to avoid disclosing any information to a potential intruder.

Return the `auto_discovered` field in the full node representation. Update the node listing (`GET /v1/nodes`) with an ability to filter by this field.

Lookup process

The lookup process is somewhat more complicated than the normal Ironic lookup because the inspected nodes may not have any ports enrolled. The procedure will try to find one and *only one* node that satisfies the provided node UUID, MAC addresses and BMC addresses.

BMC addresses are not indexed in the database and require some pre-processing. When *starting* inspection, BMC addresses will be collected from the nodes `driver_info`, resolved into IP addresses and cached in the `driver_internal_info`. On lookup, `driver_internal_info` from all nodes in the `INSPECT WAIT` state will be checked.

If none or several nodes match the data, HTTP 404 with no explanation will be returned. Extensive logging will be provided for debugging purposes.

Auto-discovery

If auto-discovery is enabled (see *auto-discovery configuration*), the lookup process will work a bit differently for completely new nodes. If no node at all can be found for data (as opposed to a node in an invalid state), a new node will be created by the API layer. The rest of inspection happens the same way.

Nodes created this way will have an `auto_discovered` field set to `True`.

Client (CLI) impact

All new API features will need to be exposed.

openstack baremetal CLI

Starting and aborting inspection.

No changes here, use the same commands:

```
$ openstack baremetal node set --inspect-interface agent <node>
$ openstack baremetal node inspect <node>
$ openstack baremetal node abort <node>
```

Exposing inspection data.

I would like to have command to display certain parts of the inventory, filter it, etc. It is unclear if we should do it on the client or server side (or not do it at all). Inspector has a couple of comments for extracting parts of the inventory. I suggest not to migrate them in the first iteration.

Well start with migrating the most basic command, simply saving the complete JSON to a file or displaying it:

```
$ openstack baremetal node inventory save [--file <file path>] <node>
```

The callback endpoint will not be exposed via the CLI.

Filtering auto-discovered nodes

Can be useful for auditing purposes, especially together with filtering on provision state:

```
$ openstack baremetal node list --provision-state enroll --auto-discovered
```

openstacksdk

Expose a call to fetch inventory, very similar to the existing call for Inspector:

```
def get_inventory(self, node):
    """Get inventory for the node.

    :param node: The value can be the name or ID of a node or a
        :class:`~openstack.baremetal.v1.node.Node` instance.
    :returns: inspection data from the most recent successful run.
    :rtype: dict
    """
```

Update the node API with filtering on `auto_discovered`.

RPC API impact

A new RPC call will be introduced for handling the inspection data:

```
def continue_inspection(self, context, node_id, inventory,
                        plugin_data=None):
    """Continue in-band inspection.

    :param context: request context.
    :param node_id: node ID or UUID.
    :param inventory: hardware inventory from the node.
    :param plugin_data: optional plugin-specific data.
    :raises: NodeLocked if node is locked by another conductor.
    :raises: NotFound if node is in invalid state.
    """
```

On receiving this call, the conductor will acquire an exclusive lock, double-check the provision state and launch a thread for further processing.

See *PXE filter script* for further impact.

Driver API impact

Extend the *inspect interface* with an additional call:

```
def continue_inspection(self, task, inventory, plugin_data=None):
    """Continue in-band hardware inspection.

    Should not be implemented for purely out-of-band implementations.

    :param task: a task from TaskManager.
    :param inventory: hardware inventory from the node.
```

(continues on next page)

(continued from previous page)

```
    :param plugin_data: optional plugin-specific data.
    :raises: UnsupportedDriverExtension, if the method is not implemented
             by specific inspect interface.
    """
```

Inspection hooks

Inspection hooks are a new kind of Ironic plugins, closely based on the Inspectors processing hooks (see the *Glossary*).

The current Inspectors processing hook interface looks like this (shortening docstrings for readability):

```
class ProcessingHook(object, metaclass=abc.ABCMeta):

    dependencies = []
    """An ordered list of hooks that must be enabled before this one."""

    def before_processing(self, introspection_data, **kwargs):
        """Hook to run before any other data processing."""

    def before_update(self, introspection_data, node_info, **kwargs):
        """Hook to run before Ironic node update."""
```

Adapting to the Ironic terminology, new API and internal structures, this becomes:

```
class InspectionHook(metaclass=abc.ABCMeta):

    dependencies = []
    """An ordered list of hooks that must be enabled before this one."""

    def preprocess(self, task, inventory, plugin_data):
        """Hook to run before the main inspection data processing."""

    def __call__(self, task, inventory, plugin_data):
        """Hook to run to process the inspection data."""
```

Hooks

- **must** override `__call__` and *may* override the other two methods.
- are always run with an exclusive lock with the node in the `INSPECTING` provision state.
- *may* modify the plugin data but *should not* modify the inventory.
- *should avoid* permanently modifying the node or any related resources in the `preprocess` phase.
- **must** call `task.node.save()` explicitly on modifications.

The ordered list of hooks that will run by default (see *hooks configuration*):

`ramdisk-error`
: Fails the inspection early if an error message is passed along the inspection data.

`architecture`
: Sets the `cpu_arch` property based on the inventory.

`validate_interfaces`
: Validates interfaces in the inventory. Valid interfaces are stored in `plugin_data` in the new key `valid_interfaces` with an additional field `pxe_enabled`.

`ports`
: Creates ports based on the `add_ports`/`keep_ports` options (see *port creation configuration*). Requires the `validate_interfaces` hook. Updates the `valid_interfaces` collection with a new boolean interface field `is_added`.

The list of available optional hooks (adapted from existing Inspector hooks):

`accelerators`
: Sets the `accelerators` property based on the available accelerator devices and the configuration.

`boot-mode`
: Sets the `boot_mode` capability based on the boot mode during the ramdisk run.

`cpu-capabilities`
: Updates capabilities based on CPU flags from the inventory.

`extra-hardware`
: Converts the extra collected data from the format of the hardware-detect tool (list of lists) to a nested dictionary. Removes the original `data` field from the `plugin_data` and creates a new field `extra` instead.

`local-link-connection`
: Uses LLDP information to set the `local_link_connection` field on ports. Can be used together with `parse-lldp`.

`memory`
: Sets the `memory_mb` property based on the inventory.

`parse-lldp`
: Converts binary LLDP information into a readable form, which is then stored in the `plugin_data` as a new `parsed_lldp` dictionary with interface names as keys.

  https://specs.openstack.org/openstack/ironic-inspector-specs/specs/lldp-reporting.html

`pci-devices`
: Updates the nodes capabilities with PCI devices using a mapping from the configuration.

  https://specs.openstack.org/openstack/ironic-inspector-specs/specs/generic-pci-resource.html

`physical-network`
: Allows setting the ports `physical_network` field based on the CIDR mapping in the configuration.

  Can be subclassed to implement a different logic.

`raid-device`
: Uses a diff between two inspection to detect the freshly created RAID device and configure it as a root device.

> **Note**
>
> The current implementation caches devices in the nodes `extra`. We should rather fetch the old inventory for that.

`root-device`
: Uses root device hints to determine the root device and sets the `local_gb` property.

Note

Nothing will set the `cpus` property. Its not used by Nova any more and should be removed from essential properties.

Nova driver impact

Fortunately, none.

Ramdisk impact

At the first pass, there will be no changes to the ramdisk. The new callback API will be fully compatible with its counterpart in Inspector.

The follow-up change will be to make lookup and inspection mutually exclusive: if inspection (at least its synchronous lookup part) succeeds, the token and node data are returned in the response, and the lookup is not needed.

Security impact

- This change introduces one more API endpoint without authentication. Knowing either UUID, MAC address or BMC address, an intruder can receive some information as well as the agent token (if it hasnt been retrieved yet) for a node in the `INSPECT WAIT` state.

  If in-band inspection is disabled or simply not used, no nodes will ever be in the `INSPECT WAIT` state since it is not used by out-of-band inspection implementations.

Other end user impact

None?

Scalability impact

The scalability impact on a deployment with in-band inspection will probably be net positive because periodic sync-ups of Inspector with Ironic will no longer be necessary.

Having PXE filters as a separate process means that they can be scaled separately from the rest of Ironic (e.g. it may make sense to keep them in an active/standby setup, while the rest of Ironic is active/active).

Performance Impact

The expected performance impact is also positive:

- Removal of the periodic task that synchronizes inspection results from Inspector to Ironic.
- More efficient database queries on inspection lookup and in PXE filters.

Storing inventory in the database does impact its size, but it is already the case for Inspector. However, during the transition period, there will be two copies of inventory. If this becomes a problem, an operator may opt to disable the inventory storage on the Ironic side until ready to switch over completely.

Other deployer impact

Hooks configuration

New configuration options in the `[inspector]` section:

`default_hooks`
A comma-separated lists of inspection hooks that are run by default. In most cases, the operators will not modify this.

The default (somewhat conservative) hooks set will create ports and set `cpu_arch`.

`hooks`
A comma-separated lists of inspection hooks to run. Defaults to `$default_hooks`.

Note

This scheme allows easily inserting hooks in the beginning or the end of the list without hardcoding the default list, e.g.:

```
[inspector]
hooks = my-early-hook,$default_hooks,later-hook-1,later-hook-2
```

Port creation configuration

Various inspection hooks will come with their configuration. The most important is the port creation options in the `[inspector]` section:

`add_ports`
Which interfaces to enroll as ports for the node. Options:

- `all` (the default) - all valid interfaces.
- `active` - only interfaces with an IP address.
- `pxe` - only the PXE booting interface.

`keep_ports`
Which existing ports to keep.

- `all` (the default) - keep all ports, do not delete anything.
- `present` - delete all ports that do not correspond to interfaces in the inventory.
- `added` - delete all ports except for ones selected via the `add_ports` option (only makes sense if `add_ports` is not set to `all`).

Disk spacing configuration

An odd quirk of our partitioning code is that the `local_gb` field has to be smaller than the actual disk, otherwise the partitioning may fail. Inspector has been dealing it by making `local_gb` 1G smaller. This will be reflected in the following option:

`disk_partitioning_spacing`
Size in GiB to leave reserved. Defaults to 1, set to 0 to disable.

Auto-discovery configuration

The new `[auto_discovery]` section will have these options:

`enabled`
Boolean field, defaults to `False`.

`driver`
The driver to use for newly enrolled nodes. Required when the feature is enabled.

> **Note**
>
> Inspector has several options to tune the freshly created nodes. I believe that this complex logic should rather be implemented with inspection rules. The follow-up inspection rules spec will have some additions to make it easier.

PXE filter script

A new executable `ironic-pxe-filter` will be introduced to support unmanaged inspection in environments with Neutron. It will be designed to be deployed alongside a separate *dnsmasq* process. Unlike the current implementation in Inspector, the script will have direct access to the Ironic database for efficiency.

Operators that do not need PXE filtering, e.g. because they only use managed inspection or use a single PXE environment (without Neutron), can opt out of running `ironic-pxe-filter`. This applies, for example, to Bifrost and Metal3.

The only downside of this approach is knowing when inspection starts or finishes. Inspector learns it immediately and is able to update the filters without a further delay. The periodic task approach will result in a delay that is probably acceptable for real hardware but will be problematic for virtual machines.

To overcome this limitation, the new executable will feature an RPC service when the RPC transport is *oslo.messaging*. This service will use a separate *topic* `ironic.pxe_filter` and will receive broadcast messages from the conductor handling inspection. The RPC call will be:

```
def update_pxe_filters(self, context, allow=None, deny=None):
    """Update the PXE filter with the given addresses.

    Modifies the allowlist and the denylist with the given addresses.
    The state of addresses that are not mentioned does not change.

    :param allow: MAC addresses to enable in the filters.
    :param deny: MAC addresses to disable in the filters.
    """
```

No notifications will be done for JSON RPC transport. This will be documented as a known limitation. The further work as part of the cross-conductor RPC effort may eventually lift it.

Developer impact

While other in-band inspection implementations are possible, theyll probably happen as downstream modifications to the proposed implementation.

3.12.3 Implementation

Assignee(s)

Primary assignee:
Dmitry Tantsur (IRC: dtantsur, dtantsur@protonmail.com)

Other contributors:
Jakub Jelínek (IRC: kubajj) - inventory API

Work Items

Too many to mention - see tasks in https://storyboard.openstack.org/#!/story/2010275.

3.12.4 Dependencies

None so far.

3.12.5 Testing

- Bifrost will be migrated to the new implementation as early as possible.
- DevStack CI coverage will be added, possibly in form of tests in standalone jobs.
- Eventually, the existing Inspector job will be migrated over or deleted.

3.12.6 Upgrades and Backwards Compatibility

Other than the eventual deprecation of Inspector itself and the corresponding *inspect interface*, the change is backward compatible on the Ironic side.

Migration will be reasonably easy, but not necessarily friction-free. Possible concerns:

- No migration for inspection data. We could provide a tool, Im just not sure if its worth the effort. Can be done as an afterthought.
- Co-existence of the new and old *inspect interfaces*.
  - The callback API will be designed to work with the old interface by proxying the data to Inspector. This way, an operator can use the same callback URL for both implementations.
  - The new PXE filters script will also function the same way for both implementations.
  - The inventory API will be implemented for the old implementation by fetching the data from Inspector on successful inspection.

3.12.7 Documentation Impact

A lot of documentation has to be written, or rather adapted from ironic-inspector. API reference will be added for all new API endpoints.

3.12.8 References

3.13 Hardware that cannot be powered off

https://bugs.launchpad.net/ironic/+bug/2077432

3.13.1 Problem description

Power off is a very fundamental action for bare-metal provisioning. Not only is it available as an API primitive, Ironic also often uses a sequence of power off and power on instead of a single reboot action. This happens for three reasons:

- By waiting for the machine to power off first, we ensure that the power off request actually came through. Some IPMI implementation are notorious for ignoring power requests under certain conditions.
- Some actions require the machine to be off to work correctly or at all. For example, some hardware was reported to refuse to mount virtual media devices on a powered on machine.
- When multi-tenant networking is used, its essentially to switch the networking with the machine powered off, otherwise the code running on it will be exposed to both networks (e.g. IPA will stay running on the tenant network).

Unfortunately, in some cases powering off a machine is not possible:

- Some implementations of the NC-SI technology suffer from a serious drawback: the NIC that is shared with the BMC is powered off whenever the machine is powered off. When that happens, it is no longer possible to power on the machine remotely.
- DPUs may not support the power off action at all, relying on the parent machine power state instead. They may support reboot/reset though.

While the second case is related to an emerging technology, the first case is already seen in the wild and causes issues with the adoption of Ironic.

3.13.2 Proposed change

Add a new optional node flag `disable_power_off`. When set to `True`, Ironic will avoid ever issuing an explicit power off request. Specifically:

- All *power interfaces* will use an explicit reboot request (or fail if its not available) even when normally they would use a power-off/power-on pair (e.g. `redfish`). To detect the reboot, well insert a hardcoded sleep and then wait for the machine to be on.
- The `tear_down_agent` deploy step will no longer try to power off the machine. Instead, after collecting the ramdisk logs, it will issue the new `lockdown` IPA command to disable IPA (see *Ramdisk impact*).
- The `boot_instance` deploy step will unconditionally use hard out-of-band reset instead of the in-band power off command. The previously issued `lockdown` command will ensure that the disk caches are flushed already.
- The `tear_down_inband_cleaning` function will issue a reboot request after de-configuring IPA via `clean_up_ramdisk`. Same for `tear_down_inband_service`.
- On deployment, cleaning, inspection or servicing failure, the machine will stay on with IPA running.
- Validation will fail for any requested deploy, clean or service steps that include an explicit power off command.

Downsides

- The usage of `disable_power_off` opens up a potential vulnerability in the multi-tenant networking because IPA will be available on the tenant network for a short time. To mitigate this problem, a new *lockdown mode* will be introduced to IPA to ensure its at least not operational any more - see *Ramdisk impact*.
- Similarly, during cleaning the instance operating system will be switched to the cleaning network before IPA boots. Well document this as a potential issue and add a new configuration option to enable the no-power-off mode together with the `neutron` interface.
- After `tear_down_agent`, the machine will still be running. Any custom deploy steps or out-of-band actions that rely on the machine to be powered off after this step may fail.
- In case of IPMI, if the BMC ignores the reboot request, well still mark it as successful.

Alternatives

Short of convincing the vendors to fix the NC-SI issue in hardware, I do not see any alternatives. The NC-SI setup seems to be gaining popularity, especially in *far edge* setups.

The first version of this specification suggested adding `disable_power_off` to `driver_info` instead of making it a first-class node field. I changed it for two reasons: because of how many unrelated places in Ironic will need to check this field and to provide an easy way to guard access to it via RBAC.

Data model impact

None

State Machine Impact

None

REST API impact

A new field will be possible to set on node creation and later on. The access to it will be guarded by a new microversion and a new RBAC rule.

Client (CLI) impact

openstack baremetal CLI

Expose the new field as `--disable-power-off` to the create/set commands.

openstacksdk

Expose the new field on the Node object.

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

Add a new command `lockdown` that will prepare the machine for a hard reset and make sure IPA is not practically usable on a running ramdisk. Namely:

- Issue `sync` and write 3 to `/proc/sys/vm/drop_caches` to flush Linux caches.
- For each device, issue `blockdev --flushbufs <device>` to flush any outstanding I/O operations.
- Stop the heartbeater thread and the API.
- Try to disable networking by running `ip link set <interface> down` for each network interface.

Security impact

See *Downsides* for security trade-offs that need to be made.

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

A new security parameter will be added:

`[neutron]allow_disabling_power_off` (boolean, default `False`)
: If `False`, the validation of the `neutron` network interface will fail for nodes that have `disable_power_off` enabled. If set to `True`, this feature will be usable together.

Developer impact

Authors of 3rd party power interfaces must take the new flag into account. Well give them a heads-up via release notes.

3.13.3 Implementation

Assignee(s)

Primary assignee:
: Dmitry Tantsur (dtantsur)

Other contributors:
TBD

Work Items

- Update all power interfaces to respect the new flag.
- Update the agent deploy steps to respect the new flag.

3.13.4 Dependencies

None

3.13.5 Testing

Its possible to add a standalone job that tests the new mode of operation. We could even modify sushy-tools to reject power-off calls, but using this approach in the CI would require a new job, and were trying to avoid new jobs.

3.13.6 Upgrades and Backwards Compatibility

No concerns

3.13.7 Documentation Impact

Add a documentation page that lists the use cases and highlights the drawbacks.

3.13.8 References

3.14 No IPA to conductor communication

https://storyboard.openstack.org/#!/story/1526486

This spec intends to make agent->ironic communication optional, instead using polling to make all communication inbound to the agent.

3.14.1 Problem description

As part of the boot process IPA must call the ironic API to query its node ID, and notify ironic when its completed the boot process.

This implies that a target node has network access to the ironic API, which means that a malicious party could in theory attack the control plane from an instance. If the same control plane holds Keystone, Neutron, or other such services, the attacker can now DoS or compromise those. This grants them significant control over the infrastructure.

A deployer could mitigate this security flaw by using two networks for hosts:

- A provisioning network, which has access to the ironic API, but no ability to communicate with other nodes in the datacenter.
- A tenant network, which can communicate with the outside world, and other hosts, but cannot contact the ironic API.

However, this doesnt scale with medium to mega scale deployers who leverage layer 3 network topologies. In L3 networks a subnet is constrained to a single rack. This means that to leverage two networks to image hosts one would need to provision a second subnet for every single rack.

Compounding the issue further, different networks with different fundamental security policies implies that these disparate policies must be enforced. Thus, for each of your provisioning networks that require access to the ironic API, there must be access controls configured and enforced in the firewall.

In the context of hundreds, or thousands of racks, this does not scale.

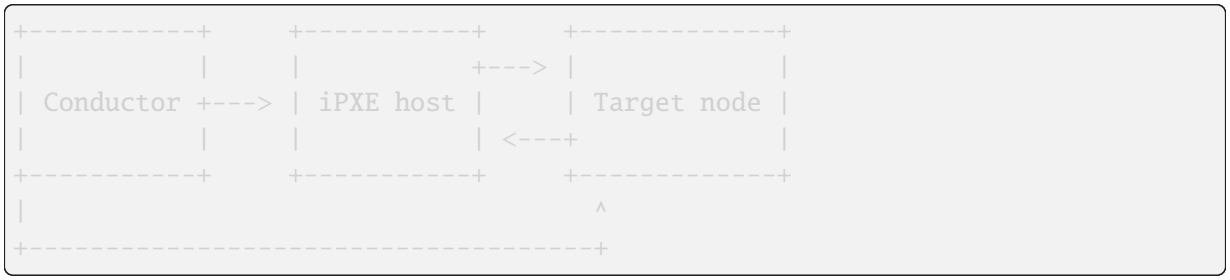
An example of the (potential) security problem:

- Bob boots a host with ironic, this host is publicly routable to the internet. Mallory finds and compromises this host. She then attacks the ironic API from this host. Once she compromises the ironic API, she then starts booting other hosts in the datacenter with a compromised disk image. If Bob uses ironic to manage every host in the datacenter, then Mallory has now effectively owned an entire datacenter.

To remediate this, we need to reduce the attack surface of the control plane by removing the need for the data plane to be able to send traffic to the API host. To do this, we need to be able to tell the agent that it should not call ironic, and make ironic poll the agent instead.

If the deployer runs DHCPD and serves PXE/iPXE from the control plane, then there is still logical network access between the target node and the control plane. However, this is easily fixed by running those services on dedicated intermediary hosts which do not have network access to the rest of the control plane.

In this context the network flow is:



And a simple example of the provisioning process would be:

1. Conductor receives call to boot node.
2. Conductor creates boot data, without the ipa-api-url parameter.
3. Conductor sends OOB call to power target node on.
4. Target node boots, uses DHCP to get IP.
5. Target node runs PXE/iPXE, pulls data from ipxe host.
6. Target boots and runs IPA.
7. Conductor polls for IPA until it is alive.
8. Conductor calls IPAs `get_hardware_info` command to get information about the nodes hardware. This is used to validate the MAC addresses to ensure this is the node we are expecting.
9. Conductor calls IPAs `node_info` command to give it the data it needs to do its job, including the config data returned by the lookup API.
10. Conductor calls IPA commands on target node to walk it through the provisioning process.
11. Conductor polls instance at a configurable interval to check on state, gather information, etc.
12. Target is complete. Conductor reboots target node.

13. Conductor cleans up boot data.
14. Complete.

An example of this as a fix to the security problem:

- Bob boots a host with ironic, this host is publicly routable to the internet. Mallory finds and compromises this host. She attempts to attack the ironic API. The connection times out. She gives up and attacks the iPXE host. She succeeds and compromises the iPXE host. She then attempts to attack the ironic API. The connection fails. Mallory, disappointed, gives up and puts her life of crime behind her.

In this case, even though Mallory has compromised the target node, there is no intrinsic network access between the target node and the control plane. Thus her only route of attack against the provisioning infrastructure would be DoS, or to impact hosts which are in the process of booting. But she has no ability to attack the queue, conductor, api, db, etc. She cannot gain control over the infrastructure, and her attack has been limited.

3.14.2 Proposed change

We will add two options to the [agent] category:

- `poll_only`: BoolOpt to enable passive mode. Defaults to False.
- `poll_interval`: IntOpt, poll interval in seconds. Defaults to the current `[api]/ramdisk_heartbeat_timeout` setting.

And one option to the [api] category:

- `disable_agent_api`: BoolOpt which disables the agent lookup and heartbeat APIs. Defaults to False.

If `poll_only` is enabled, we do not pass the `ipa-api-url` kernel command line parameter to IPA, which will disable the node lookup and heartbeat mechanisms.

If `poll_only` is enabled, the conductor will use a periodic task to query each agent at an interval as defined in poll_interval instead of querying the agent after a heartbeat is received. This periodic task will only query nodes in states IPA would normally be heartbeating in: (DEPLOY*, RESCUE*, CLEAN*).

It is assumed that the deployer should disallow communication between the target node and the ironic API. However, if an API call does come through when `disable_agent_api` is True, then Ironic should return a 403.

For this mode, we will also need to remove `ipa-api-url` being passed as kernel parameter to the agent.

We will also add a `node_info` command to IPA, described below, which the conductor will use to pass the lookup data to a node.

Last, we will add a `get_hardware_info` command to IPA, which will return hardware info we can use to ensure the node is the node we are expecting.

Notes:

- This spec depends on the assumption that ironic can look up the node IP in Neutron. Deployments without Neutron are not supported with poll_only=True. This may be added in the future.
- ironic-inspector is out of scope for this feature, as it does not use Neutron.
- There may be a use case to set `poll_only` per node, rather than globally. However, this is outside the scope of this spec.

Alternatives

None.

Data model impact

None

State Machine Impact

None

REST API impact

The lookup and heartbeat APIs used by agents will now return a 403 when `disable_agent_api` is set to True.

Client (CLI) impact

None

ironic CLI

None

openstack baremetal CLI

None

RPC API impact

None

Driver API impact

Deploy drivers will need to ensure that anything reaching into some agent can also be triggered by a periodic task.

Nova driver impact

None

Ramdisk impact

ironic-python-agent mostly supports this already, as it will run just fine without an API URL.

Some steps may also require other data returned by the lookup endpoint. Well add a new synchronous command `node_info`, which will take this data as a single `node_info` argument and store it in memory for later use. Ironic will call this command when it first notices that IPA is up.

To validate the node is the node we expect, well add another synchronous command `get_hardware_info`. This will return the MAC addresses at first, but could be evolved later to include things like serial numbers, etc.

Security impact

This change will prevent a malicious actor from using IPA as a vector of attack against the ironic API.

Note that TLS on the agent API is still important to completely secure the interactions between IPA and Ironic; however, this is outside the scope of this spec.

Other end user impact

None

Scalability impact

Polling target nodes for state from the conductor could have scale issues when managing many thousands of nodes. However, polling will be done in a thread pool, and so there should be limited impact.

Performance Impact

Polling in a large parallel fashion will introduce additional CPU load on the conductor nodes. Deployers may need to scale out their conductor nodes to handle the additional load.

Other deployer impact

Recap of the configuration options added:

[agent] * poll_only (type=BoolOpt, default=False) * poll_interval (type=IntOpt, default=<[api]/ramdisk_heartbeat_timeout>)

[api] * disable_agent_api (type=BoolOpt, default=False)

We should document where each of these needs to be set (API vs conductor hosts).

Developer impact

None

3.14.3 Implementation

Assignee(s)

Primary assignee:
jroll

Other contributors:
penick

Work Items

- Enable IPA to skip the lookup process when ironic does not pass the `ipa-api-url` kernel parameter.
- Create the `get_hardware_info` IPA command.
- Create the `node_info` IPA command.
- Add the new options to ironic.
- Enable Ironic to use polling for agent actions/status rather than using the heartbeat as a trigger.
- Make ironic call the `node_info` command after IPA boots, when in polling mode.

- Disable heartbeating in the agent in polling mode.
- Test scale and performance impact on periodic tasks.
- Lots of documentation, especially in admin guides. It may also be worth a large blurb in the reference architecture guide.

3.14.4 Dependencies

None.

3.14.5 Testing

We should configure one of the existing tempest jobs to use this feature.

3.14.6 Upgrades and Backwards Compatibility

The deployer must update IPA in their images to support passive mode prior to upgrading Ironic and enabling the feature. If they do not, all imaging attempts will fail.

3.14.7 Documentation Impact

This feature needs to be documented as a deployment option.

The ironic-inspector docs need to be updated to capture that inspector wont work with poll_only=True.

Admin docs should be updated to note that firewall rules need to be implemented to actually close network access between the target node and the ironic API.

3.14.8 References

None

3.15 Nodes tagging

https://bugs.launchpad.net/ironic/+bug/1526266

This aims to add support for tagging nodes.

3.15.1 Problem description

Ironic should have tags field for every node, which can be used to divide the nodes to some groups. then we can do list by tag to get a group of nodes with same properties like hardware specs.

3.15.2 Proposed change

- Add APIs that allows a user to add, remove, and list tags for a node.
- Add tag filter parameter to node list API to allow searching for nodes based on one or more string tags.

Alternatives

Current chassis object is kind of an alternative for grouping nodes.

Data model impact

A new *ironic.objects.tags.NodeTagList* object would be added to the object model.

The *ironic.objects.tags.NodeTagList* field in the python object model will be populated on-demand (i.e. not eager-loaded).

A tag should be defined as a Unicode string no longer than 255 characters in length, with an index on this field.

Tags are strings attached to an entity with the purpose of classification into groups. To simplify requests that specify lists of tags, the comma character is not allowed to be in a tag name.

For the database schema, the following table constructs would suffice

```
CREATE TABLE node_tags (
    node_id INT(11) NOT NULL,
    tag VARCHAR(255) CHARACTER SET utf8 NOT NULL,
    PRIMARY KEY (node_id, tag),
    KEY (tag),
    FOREIGN KEY (node_id)
      REFERENCES nodes(id)
      ON DELETE CASCADE,
)
```

REST API impact

We will follow the API Working Groups specification for tagging, rather than invent our own.

Will support addressing individual tags.

RPC API impact

None

State Machine Impact

None

Client (CLI) impact

Add tags CRUD operations commands:

- ironic node-tag-list <node uuid>
- ironic node-tag-update <node uuid> <op> <tags>

<op> Operation: add or remove

For individual tag: * ironic node-tag-add <node uuid> <tag> * ironic node-tag-remove <node uuid> <tag>

Add tag-list filtering support to node-list command:

- ironic node-list tag tag1 tag tag2
- ironic node-list tag-any tag1 tag-any tag2
- ironic node-list not-tag tag3

Multiple tag will be used to filter results in an AND expression, and tag-any for OR expression, allowing for exclusionary tags via the not-tag option.

Driver API impact

None

Nova driver impact

The tags information can be used for nova but its not being considered as part of this spec, and may be addressed at a later time.

Ramdisk impact

N/A

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

None

Developer impact

None

3.15.3 Implementation

Assignee(s)

Primary assignee:
niu-zglinux

Work Items

- Add *node_tags* table with a migration.
- Add DB API layer for CRUD operations on node tags.
- Added DB API layer for node tag-list filtering support.
- Add NodeTag, NodeTagList objects and a new tags field to Node object.
- Add REST API for CRUD operations on node tags.

- Add REST API for node tag-list filtering support.
- python-ironicclient additions and modifications.

3.15.4 Dependencies

None

3.15.5 Testing

Add unit tests. Add tempest API tests.

3.15.6 Upgrades and Backwards Compatibility

Add a migration script for DB.

3.15.7 Documentation Impact

Ironic API and python-ironicclient will need to be updated to accompany this change.

3.15.8 References

1. http://specs.openstack.org/openstack/api-wg/guidelines/tags.html

3.16 Override PXE options via Glance property

https://bugs.launchpad.net/ironic/+bug/1526409

The proposal presents the work required to allow PXE boot an image using a specific kernel command line for that image.

3.16.1 Problem description

Some images requires having specific kernel command line to boot correctly. Currently, Ironic tries to determine things like the root filesystem the image is on (root= kernel command line) by getting the UUID of the image filesystem and setting as the root= parameter. This is not correct for all images because some may need to boot from a different root filesystem, e.g:

- The device path of its LVM volume [root=/dev/mapper/vg-lv_root]
- From one squashfs filesystem [root=live:<path>]
- A btrfs subvolume [root=/dev/disk/by-uuid/<UUID of btrfs-root>]

As a real example, the Fedora Atomic uses `lvm` and `ostree`. It requires a kernel command line as the example below to boot properly:

```
nofb nomodeset vga=normal console=tty1 no_timer_check
rd.lvm.lv=atomicos/root root=/dev/mapper/atomicos-root
ostree=/ostree/boot.0/fedora-atomic/a002a2c2e44240db614e09e82c/0
```

3.16.2 Proposed change

In the same way Ironic look at the Glance image properties to find out the Glance UUID for the images kernel and ramdisk (kernel_id and ramdisk_id) and populate the nodes `instance_info` field with the `kernel` and `ramdisk` keys.

This spec propose having a new images property field called `kernel_cmdline` that Ironic will look at, and if present, it will populate the nodes `instance_info` field with a `kernel_cmdline` key.

If the `instance_info.kernel_cmdline` is populate Ironic will skip getting the root partition filesystem UUID and will use the command line from the `instance_info` field instead of the default one in the template. Its important to note that the values present at the `pxe_append_params` configuration option will still be appended at the end of the images custom kernel command line by Ironic.

Setting the keys in the `instance_info` field is important because it also allows Ironic to operate in standalone mode without Glance.

Alternatives

Always use whole disk images when deploying images that requires a specific kernel command line

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

N/A

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

Deployers that wants to deploy an image with a specific kernel command line should know and set it in Glance images property prior to trying to boot the image.

Developer impact

None

3.16.3 Implementation

Assignee(s)

Primary assignee:
lucasagomes <lucasagomes@gmail.com>

Other contributors:
None

Work Items

- Make Ironic look at the `kernel_cmdline` image property in Glance and if present set it to the nodes `instance_info` field
- When preparing to boot the users image, make Ironic check if the nodes `instance_info` field contains a key called `kernel_cmdline` (along with `kernel_id` and `ramdisk_id`) and if so, use that kernel command line to boot the image.

3.16.4 Dependencies

None

3.16.5 Testing

- Unit Tests

3.16.6 Upgrades and Backwards Compatibility

None

3.16.7 Documentation Impact

The Ironic deploy documentation will be updated to reflect the changes made by this spec.

3.16.8 References

- Kernel parameters
- Project atomic

3.17 Self-Service via Runbooks

https://bugs.launchpad.net/ironic/+bug/2027690

With the addition of service steps, combined with owner/lessee, we now have an opportunity to allow project members to self-serve many maintenance items by permitting them access to curated runbooks of steps.

This feature will primarily involve extending creating a new runbook concept, allowing lists of steps to be created, associated with a node via traits. These runbooks will then be able to be used in lieu of a list of steps when performing manual cleaning or node servicing.

3.17.1 Problem description

Currently, users of the Ironic API as a project-scoped member have limited ability to self-serve maintenance items. Ironic operators are given the difficult choice of giving users broad access to nodes, allowing them to run arbitrary manual cleaning or service steps with the only alternative being permitting no access to self-serve these maintenance items.

Use cases include:

- As a project member, I can execute runbooks via Node Servicing without granting the ability to execute arbitrary steps on a node.
- As an system manager, I want to store a list of steps to perform an action in an identical manner across many similar nodes.

3.17.2 Proposed change

The proposed change is to create a new API concept, runbooks, which can be used with any API flow which currently takes explicit lists of steps.

Those runbooks can then be used instead of a list of clean_steps or service_steps[0] when setting node provision state. These are expected to behave identical to API calls with `clean_steps` or `service_steps` provided, including honoring the `disable_ramdisk` field, and providing explicit ordering rather than the priority-based ordering that is used in automated cleaning and deploys.

Additionally, we will ensure that the full CRUD lifecycle of runbooks is made role-aware in the code, so that a project can limit who can create, delete, edit, or mark runbooks as public all as separate policy toggles. We will also ensure deployers can separately toggle the ability to run step-based flows via runbooks versus step-based flows with arbitrary step lists.

A runbook will only run on a node who has a trait equal to the runbook name, to ensure the runbook has been approved for use on a given piece of hardware, as an extra precaution against hardware breakage.

[0] *Change Node Provision State*: https://docs.openstack.org/api-ref/baremetal/#change-node-provision-state

Alternatives

We considered, originally, repurposing the existing deploy templates into a generic concept of templates. This was abandoned due to deploy templates containing implicit steps, making it difficult to reason about them. This is why we instead chose to call them runbooks, which are entirely specified as opposed to templates, which are partially specified and have implicit steps integrated.

Data model impact

Create new tables described below:

```
``runbooks`` (same as ``deploy_templates`` except addition of ``owner`` and␣
↪``public``)
  - id (int, pkey)
  - uuid
  - name (string 255)
  - public (bool) - When true, template is available for use by any project.
  - owner (nullable string, usually a keystone project ID)
  - disable_ramdisk - When true, similar behavior to disable_ramdisk in␣
↪manual cleaning -- do not boot IPA
  - extra json/string
  - steps list of ids pointing to ``runbook_steps``

``runbook_steps``
  - id (int, pkey)
  - runbook_id (Foreign Key to runbooks.id)
  - interface
  - step
  - args
  - order (or some other field/method to indicate how the steps were ordered␣
↪coming into the API)
```

Note: Ensure all queries to `runbooks` only pull in `runbook_steps` if needed.

State Machine Impact

While no states or state transitions are being proposed, the APIs to invoke some of those state transitions will need to change to become runbook-aware.

REST API impact

A new top level REST API endpoint, `/v1/runbooks/` will be added, with basic CRUD support.

The existing `/v1/nodes/<node>/states/provision` API will be changed to accept a `runbook` (name or uuid) in lieu of `clean_steps` when being used for servicing or manual cleaning.

Client (CLI) impact

The CLI will be updated to add support for the new API endpoints.

Some examples of CLI commands that will be added, and how they interact with RBAC:

```
- baremetal runbook create X [opts] # as system-scoped manager
  - owner: null
```

(continues on next page)

(continued from previous page)

```
  - public: false
- baremetal runbook create X [opts] # as project-scoped manager
  - owner: projectX
  - public: false
- baremetal runbook set X --public # as system-scoped manager
  - owner: null
  - public: true
  - Note: Owner field is nulled even if it previously set.
- baremetal runbook set X --public # as project-scoped manager
  - Forbidden! Requires system-scoped access.
- baremetal runbook unset X --public # as system-scoped manager
  - owner: null
  - public: false
- baremetal runbook set X --owner projectX # as system-scoped manager
  - owner: projectX
  - public: false
  - Note: Will return an error if ``runbook.public`` is true.
- baremetal node service N --runbook X
- baremetal node clean N --runbook X
- baremetal node service N --runbook X --service-steps {} # NOT PERMITTED
- baremetal node clean N --runbook X --clean-steps {} # NOT PERMITTED
```

RPC API impact

RPC API will be modified to support runbooks in lieu of steps where necessary. They will be properly versioned to ensure a smooth upgrade.

Driver API impact

None

Nova driver impact

None

Ramdisk impact

None

Security impact

Operators are warned that even with use of this feature, users may be able to leverage steps or access which are innocuous on their own, but malicious when combined.

Deployers should ensure they have reviewed all possible threat models when granting additional access to less-trusted individuals including restricting unsafe node actions, such as replacing `deploy_ramdisk` to ensure runbooks (and other step-based workflows) operate as expected.

Things for the implementer to avoid to ensure secure implementation:

- Do not permit a project-scoped API user to change `runbooks.public` by default.
- Do not permit a project-scoped API user change `runbooks.owner` by default.

- Anything that would *implicitly* mark a runbook as non-public.
- Ensure we check if nodes are able to run a given runbook using node traits, in a similar method to how we do so with deploy templates.

RBAC Impact

There are two primary ways this feature interacts with RBAC, beyond the obvious CRUD for runbooks.

First, the `runbooks.owner` and `runbooks.public` fields are relevant for determining if a runbook is scoped to a project or to a system. If `owner` is non-null and `public` is false, the runbook is scoped to the project set in that field and is only usable on nodes owned or leased by that project. If `owner` is null and `public` is false, the runbook is only able to be used or access by system-scoped users. If `owner` is null and `public` is true, a system-scoped member can modify the runbook and a project-scoped member could use it on a compatible node. Additionally, the `owner` field will only be settable when `public` is false or being set to false, and setting `public` to true will null the owner field.

Second, the node change provision state[Page 155, 0] API will have a `runbook` field added, and policy will be different for cases where `runbook` is specified instead of `clean_steps`. Default policy will be to permit manual cleaning and servicing for a node owner or lessee-scoped member when using a runbook, but to disallow it when specifying `clean_steps`. Combining `clean_steps` and `runbook` will not be permitted.

Expected access after this implementation is complete:

```
System
- Admin
- Manager
- Member
--> Can CRUD system-scoped templates (template.owner=null)
--> Can CRUD project-scoped templates (template.owner=PROJECT)
--> Can unset template.owner, changing a template to system-scope
--> Can mark system-scoped templates as public (template.public=True)
- Reader
--> Can list all templates

Project
- Admin
- Manager
--> Can CRUD project-scoped templates (template.owner=PROJECT)
--> Cannot set a template to public (template.public=True).
- Member
--> Can execute public templates or templates owned by their project.
- Reader
--> Can list public templates and templates owned by their project.
```

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

None

Developer impact

None

3.17.3 Implementation

Assignee(s)

Primary assignee:
JayF <jay@jvf.cc>

Other contributors:
TheJulia <juliaashleykreger@gmail.com>

Work Items

- Create Runbooks - Object layer - DB layer - API layer
- Add policy checking tests for /v1/runbooks
- Ensure tempest API tests exist for new API endpoints
- Update API-Ref
- Update Manual Cleaning and Node Servicing documentation

3.17.4 Dependencies

All dependencies have been resolved.

3.17.5 Testing

Unit tests will be added to test the new functionality. Integration tests will be added to test the new API endpoints and CLI commands.

3.17.6 Upgrades and Backwards Compatibility

The changes are backwards compatible. Existing API endpoints will continue to function as before, and we will gate all API changes behind microversion checks.

3.17.7 Documentation Impact

The new functionality will need to be documented. This includes documentation for the new API endpoints and CLI commands, as well as documenting security caveats detailed above.

3.17.8 References

3.18 Synchronize events with Neutron

https://storyboard.openstack.org/#!/story/1304673

Most of neutron operations are asynchronous. We need to keep track of their results by leveraging neutrons notifications, that it can send on certain events to ironic via HTTP calls.

3.18.1 Problem description

Updates to Neutron resources via its API are processed asynchronously on its backend. This exposes potential races with Ironic.

Example: an API request from Ironic to update a ports DHCP settings will return successfully long before the associated dnsmasq config has been updated and the server restarted. There is a potential for a race condition where ironic will boot a machine before its DHCP has been properly configured, especially if the machine boots very quickly (e.g. a local VM).

Another issue, that has a more serious security impact, is if Neutron failed to bind the port when configuring the tenant network, ironic proceeds with finishing the deployment, leaving the port bound to the provisioning network.

Ironic should be able to receive notifications from neutron when the port state is changed. Only Ironic related ports should cause neutron to send notifications. This spec concentrates on the binding operations, but the event handlers can be extended further, for example to be able to process updates to DHCP options. Here is an example of nova notifier in neutron[1].

In this spec, the term notification is used when talking about the notifications sent by neutron. The term event is more generic, and is used when talking about the payload that is received on the ironic HTTP API.

3.18.2 Proposed change

When ironic changes neutron port information during provisioning or cleaning (e.g. updates port binding) or creates a port (multitenancy integration), we put the node into a `*WAIT` state, and pause the deployment/cleaning waiting for neutron notification. As several ports can be updated simultaneously, we need to put the node into a `*WAIT` state only after sending the requests for all of them.

The notifications are generated on neutron side, and are sent to a dedicated ironic API endpoint - `/events`. The framework for this is already present, there is no need for additional work on neutron side. Nova notifier can be seen at[1]. An external network event will contain the following fields:

```
{
    "events": [
        {
            "event": "network.bind_port"
            "port_id": "VIF UUID",
            "mac_address": "VIF MAC address",
            "status": "VIF port status",
            "device_id": "VIF device ID",
            "binding:host_id": "hostname",
        },
```

(continues on next page)

[1] https://github.com/openstack/neutron/blob/master/neutron/notifiers/nova.py

(continued from previous page)

```
        ...
    ]
}
```

Event handler is an object processing the events received on the `/events` endpoint. This spec handles the neutron notifications case, but to make it more generic, it is proposed to define the event handlers inside the related driver interface, in the interface class `event_handler` field. For example, for network related events, it is going to be the following:

```
class NeutronNetwork(common.VIFPortIDMixin,
                     neutron.NeutronNetworkInterfaceMixin,
                     base.NetworkInterface):
    """Neutron v2 network interface"""

    event_handler = NeutronEventHandler()
```

The base `BaseNetworkEventHandler` class will contain the following methods:

```
class BaseNetworkEventHandler(object):

    @abc.abstractmethod
    def configure_tenant_networks(self, task):
        """Ensures that all tenant ports are ready to be used."""
        pass

    @abc.abstractmethod
    def unconfigure_tenant_networks(self, task):
        """Ensures that all tenant ports are down."""
        pass

    @abc.abstractmethod
    def add_provisioning_network(self, task):
        """Ensures that at least one provisioning port is active."""
        pass

    @abc.abstractmethod
    def remove_provisioning_network(self, task):
        """Ensures that all provisioning ports are deleted."""
        pass

    @abc.abstractmethod
    def add_cleaning_network(self, task):
        """Ensures that at least one cleaning port is active."""
        pass

    @abc.abstractmethod
    def remove_cleaning_network(self, task):
        """Ensures that all cleaning ports are deleted."""
        pass
```

In the conductor methods that deal with network interface (e.g., do_node_deploy), were going to be

saving the events we are expecting in the nodes `driver_internal_info['waiting_for']` field and calling the network interface methods (this will be moved to conductor from the deploy interface). If the call is synchronous, well just proceed to the next one when the previous is done, otherwise well be triggering the state machine wait event and saving the callback that will be called when the corresponding event is done. All callbacks will be stored in a simple dictionary keyed by node UUID. `driver_internal_info['waiting_for']` is going to be a simple list of strings, each of which is going to be the corresponding driver interface and event handlers method name, so that we know which method of the event handler of a specific driver interface to trigger when an action is asynchronous and we receive the event. If an unexpected event is received, well be ignoring it (and logging that an unexpected event appeared in the API).

Third-party driver interface methods can be also adding things they want to wait for by:

- adding event names into the `driver_internal_info['waiting_for']` list;
- adding event names to callback mappings into global per-node `CALLBACKS` dictionary, along with the arguments with which it should be called.

This will allow to wait for custom events registered in custom driver interfaces.

Neutron does not know which method name it should include in the request body, as it only operates on the neutron entities, it knows about things such as port bound, port unbound, port deleted etc. We will be mapping the things were waiting for to things neutron passes in via the simple dictionary:

```
NETWORK_HANDLER_TO_EVENT_MAP = {
    'network.unconfigure_tenant_networks': 'network.unbind_port',
    'network.configure_tenant_networks': 'network.bind_port',
    'network.add_provisioning_network': 'network.bind_port',
    'network.remove_provisioning_network': 'network.delete_port',
    'network.add_cleaning_network': 'network.bind_port',
    'network.remove_cleaning_network': 'network.delete_port',
}
```

When an external network event is received, and if were waiting for it, ironic API performs node-by-mac and port-by-mac lookup, to check that the respective node and port exist. The port status received in the request body is saved to the ports `internal_info['network_status']`, and then `process_event` is triggered. On the conductor side, `process_event` will be doing the event name to event handler method translation via `NETWORK_HANDLER_TO_EVENT_MAP`, and calling the event handler. Conductor will also be dealing with state machine transitions.

The event handler will be looking at the status of the ironic resources, for example, in case of network events, we want to save the neutron port status in each port or port group to `internal_info['network_status']` and consider an asynchronous action done only when port(group)s have the desired status. The event handler method that needs to be called on the event retrieval should be present in the event body generated by neutron. In case of desired event is done, it should be removed from the `driver_internal_info['waiting_for']` list, and the provisioning action can proceed, by triggering the continue state machine event and calling the callback that we have saved before.

To ensure that we dont wait for events forever, the usual `*WAIT` states timeout periodic tasks will be used. A new one will be added for the new `DELETE WAIT` state. An example of such periodic task is at[2].

[2] https://github.com/openstack/ironic/blob/f16e7cdf41701159704697775c436e9b7ffc0013/ironic/conductor/manager.py#L1458-L1479

Alternatives

- Using semaphores to pause the greenthread while waiting for events. This will make the code clearer and simpler, with only one downside if the conductor is going to be restarted, well loose the info about the events we wait for. This is still better than what we have now, and possibly can be worked around. Another downsides here being possible performance issues if a lot of greenthreads are running sumiltaneously, and the fact that conductor goes down during the rolling upgrade.
- Use Neutron port status polling. There is an issue with that, as even if the neutron ports status is ACTIVE, some actions might not have finished yet. The neutrons notifier framework handles this problem for us.

Data model impact

None.

State Machine Impact

A new `DELETE WAIT` state is introduced. Nodes can move to it from `DELETING` state, upon receiving wait event. When continue event is triggered while the node is in `DELETE WAIT`, the node switches back to the `DELETING` state. This is introduced because we need to unconfigure the tenant networks prior to starting the cleaning.

REST API impact

The new endpoint `POST /events` needs to be created. The default policy for this endpoint will be `"rule:is_admin"`. Request body format is going to be the following:

```
{
    "events": [
        {
            "event": "network.bind_port"
            "port_id": "VIF UUID",
            "mac_address": "VIF MAC address",
            "status": "VIF port status",
            "device_id": "VIF device ID",
            "binding:host_id": "hostname",
        },
        ...
    ]
}
```

Only `event` field is required, and it has the format of `<event_type>.<event>`, where:

- `<event_type>` is a name of the interface whose event handler will be called, during this spec implementation only `network` interface handlers will be added.
- `<event>` is a name of the event that has happened, it will be converted to the event handler method name of the current `<event_type>` interface handler that will be called.

If the expected event handling fails, `fail` state machine event is triggered by the conductor.

The normal response code to the request on this endpoint is 200 (OK), the error codes are:

- 400 (Bad Request), in case of none of the event handlers can process the event.

- 404 (Not Found), in case of making a request with old API microversion header, or if the node can not be found by the MAC address that is sent in the request body.
- 401 (Unauthorized), if the authorization has been refused for the provided credentials.
- 403 (Forbidden), if the user that has issued the request is not allowed to use this endpoint.

Client (CLI) impact

Client will be updated to support sending an external notification. This functionality will only be added to the clients python API, no new commands are going to be introduced. The new method will just be passing the JSON it receives to the `/events` endpoint.

This method will be used by the ironic notifier module within neutron to send the notification to the ironic API.

RPC API impact

A new method `process_event` is going to be added. Received external event is processed here.

In the conductor side of this method we compare current event were waiting for stored in `driver_internal_info['waiting_for']` field with received `"event"` in the event body. If we received the desired event for all port(group)s we need, we trigger `continue` event on the state machine, and the callback that was saved into the per-node `CALLBACKS` dictionary prior to triggering the state machines wait is called.

Driver API impact

As part of this change, to ensure that the network interface calls happen, and we wait for their completion, well need to make the `add_{cleaning,provisioning}_network` network interface methods idempotent, so that we can call them in the conductor without breaking the out-of-tree network interfaces.

Nova driver impact

None.

Security impact

With the move of the network interface calls to conductor, and waiting for their successful completion, we ensure that the network configuration corresponds to what we expect, thus enhancing security, and getting rid of bugs with giving an instance to a user that is still mapped to provisioning network if `remove_provisioning_network` and `configure_tenant_networks` methods fail asynchronously for that port.

Other end user impact

The neutron notifier needs to be configured. It needs the keystone admin credentials and (optionally, if not provided will be discovered from keystone endpoint catalog) the ironic API address to send events to.

Scalability impact

None.

Performance Impact

The node provisioning and unprovisioning may take some additional time when well be waiting for the external events.

Other deployer impact

None.

Ramdisk impact

None.

Developer impact

Developers will be able to create the needed event handlers for whatever events they would like to use during provisioning, and add those to the driver interfaces.

3.18.3 Implementation

Assignee(s)

Primary assignee:
vdrok

Other contributors:
None

Work Items

1. Add the event handlers to neutron and flat network interfaces.
2. Add the `process_event` conductor method, that will be handling the events.
3. Add the `/events` endpoint.
4. Implement the client side of changes.

3.18.4 Dependencies

- Nova should be cleaning up only the ports owned by compute in case of the InstanceDeployFailure[3].

3.18.5 Testing

Integration and unit testing will be provided.

3.18.6 Upgrades and Backwards Compatibility

This does not affect the usual upgrade procedure. To make use of events, both API and conductor need to be upgraded. During upgrade, the ironic notifier needs to be configured in neutron. There is going to be no need to enable this feature, it will be enabled by default.

In case of rolling upgrade, ironic conductors are upgraded first, then ironic APIs, then neutron is reconfigured to enable the notifier.

[3] https://bugs.launchpad.net/nova/+bug/1673429

If we decide to make all the network interface calls asynchronous, step of enabling notifier in neutron becomes obligatory, otherwise an operator will have to send the notifications to ironic API manually, or the deployment will be failing by timeout as no network event is going to be received. This bit might need to be revisisted during review :)

3.18.7 Documentation Impact

This feature will be documented in the developer documentation and API reference.

3.18.8 References

CHAPTER

FOUR

BACK-LOG OF IDEAS

These specifications are ideas and features that are desirable but do not have anyone working on them:

4.1 Introduce Driver Capabilities

This spec introduces inspectable (via Ironic API) capabilities for Ironic drivers.

https://blueprints.launchpad.net/ironic/+spec/driver-capabilities

4.1.1 Problem description

Capabilities are required for at least these use cases:

- Scheduling on a node with a proper driver.

  E.g. with time we start implementing variants of deployment that are not supported by all drivers, including:

  - Whole-disk vs partition deployment [1].
  - Using configdrive for deployment [2].
  - Using ephemeral partitions.

  We need to take all these differences into account when scheduling. Failure to do so will lead to errors on the late stage of deployment or even (as with whole-disk) to the wrong deployment.

  Ironic should be able to aggregate driver capabilities into node capabilities to expose them to the scheduler. (Not addressed by this spec.)

- Examining driver features.

  As we are introducing more vendor-specific things, we need some way to find out, if one is supported by a given driver. Driver capabilities would enable 3rd-party tools to inspect support of particular feature by particular driver.

4.1.2 Proposed change

- Introduce get_capabilities() call to all driver interfaces, return value being set of strings.
- Introduce get_capabilities() call to `BaseDriver`, defaulting to union of capabilities of all interfaces.
- Publish capabilities as `/v1/drivers/{name}/capabilities` REST endpoint.

With the above changes, a developer could use the Ironic API to go through the list of drivers and inspect their capabilities.

Alternatives

- Obviously the best solution would be to have all drivers implement the same set of capabilities. Unfortunately, its not realistic. E.g. some drivers (like IPMI) are more generic than the other (like ILO and DRAC), but we dont want to artificialy limit what more specific drivers can expose.

Data model impact

None. For now we expect driver capabilities to be a hardcoded constant.

REST API impact

- `/v1/drivers/{name}/capabilities`
  - Get driver capabilities.
  - Method type GET.
  - Normal http response code(s): 200
  - Expected error http response code(s)
    - 404 - driver not found.
  - Parameter: `name` - driver name
  - Body: None
  - Response: JSON list of strings.
- A corresponding change in the client library and CLI is necessary, e.g.

```
$ ironic driver-get-capabilities pxe_ipmitool
```

RPC API impact

- New synchronous RPC method `get_driver_capabilities`, taking `driver_name`, returning list of capabilities.

Driver API impact

- New method `get_capabilities` added to all interfaces, defaulting to returning an empty set.
- New method `get_capabilities` added to the `BaseDriver`, defaulting to returning a union of `get_capabilities` results for all interfaces.

Change is backward-compatible.

Nova driver impact

None for now. This spec is a starting point to implement hardware capabilities for nodes [3], which will affect Nova driver.

Security impact

None

Other end user impact

- New CLI command: `ironic driver-get-capabilities <driver name>`

Scalability impact

None

Performance Impact

None

Other deployer impact

None

Developer impact

- Driver developers can override `get_capabilities` to provide information about additional capabilities.

4.1.3 Implementation

Assignee(s)

Primary assignee:
None

Work Items

- Add `get_capabilities` to all interfaces.
- Add `get_capabilities` to the `BaseDriver`.
- Add `get_driver_capabilities` to the RPC API.
- Add new REST API.
- Research whether to add capabilities to existing drivers.

4.1.4 Dependencies

None

4.1.5 Testing

Unit tests

4.1.6 Upgrades and Backwards Compatibility

No upgrade impact

4.1.7 Documentation Impact

- New API should be documented.
- Driver documentation should mention the new method.

4.1.8 References

- [1] https://review.opendev.org/97150
- [2] https://review.opendev.org/99235
- [3] https://review.opendev.org/131272

4.2 BMC event framework

https://storyboard.openstack.org/#!/story/2008366

The goal of this spec is to provide an API to manage subscriptions for BMC events. The user will be able to provide a URL where the BMC will post the events.

Non-goals: * Unify event formats or payloads across drivers.

- Provide a way to poll for events.
- Proxy notifications (see [RFE 2008555](https://storyboard.openstack.org/#!/story/2008555)).
- Store events in ironic at all.
- Update an existing subscription in the BMC, at this time, as some vendors support partial updates where as other vendors essentially require a delete/re-creation to perform an update. This may be something that can be added later, but it seems not feasible at this time.
- Choosing EventTypes when creating a subscription wont be supported, since option is deprecated since [EventDestination v1_5_0](https://redfish.dmtf.org/schemas/v1/EventDestination.v1_5_0.json).
- Support for creating subscriptions with HTTP Headers.

4.2.1 Problem description

Some BMCs have support to subscribe to specific event notifications about the hardware (e.g., overheating, removal of the device).

- As an ironic user, I want to configure the BMC to send event notifications about potential failures to a specific URI.

4.2.2 Proposed change

Overview

This RFE proposes a new top level ReST API *subscriptions* that will allow listing, creating and deleting subscriptions for nodes.

Subscriptions workflow

1. Create a subscription `POST /v1/nodes/<node_ident>/management/ subscriptions`
2. Delete a subscription `DELETE /v1/nodes/<node_ident>/management/ subscriptions/ <subscription_bmc_id>`
3. List subscriptions `GET /v1/nodes/<node_ident>/management/subscriptions`
4. Show subscription `GET /v1/nodes/<node_ident>/management/subscriptions/ <subscription_bmc_id>`

Alternatives

The user can directly access the BMC and configure the subscriptions.

Data model impact

None.

State Machine Impact

None.

REST API impact

Update the REST API for the node object to allow create/delete/list event subscriptions.

- `GET /v1/nodes/<node_ident>/management/subscriptions`

  Retrieves a list of all subscriptions available. Returns a JSON object listing all available subscriptions or empty list.

  Error codes:

  - 404 - Node Not Found / microversion not high enough for API consumer.

  Example response object:

```
{
  "subscriptions": [
    {
      "id": "<subscription_bmc_id1>",
      "links": [
        {
          "href": "http://127.0.0.1:6486/v1/nodes/<node_id>/management/
          subscriptions/<subscription_bmc_id1>",
          "rel": "self"
        },
        {
          "href": "http://127.0.0.1:6486/nodes/<node_id>/management/
          subscriptions/<subscription_bmc_id1>",
          "rel": "bookmark"
        }
      ]
    },
    {
      "id": "<subscription_bmc_id2>",
      "links": [
        {
          "href": "http://127.0.0.1:6486/v1/nodes/<node_id>/management/
          subscriptions/<subscription_bmc_id2>",
          "rel": "self"
        },
        {
          "href": "http://127.0.0.1:6486/nodes/<node_id>/management/
          subscriptions/<subscription_bmc_id2>",
```

(continues on next page)

(continued from previous page)

```
            "rel": "bookmark"
          }
        ]
      },
    ]
}
```

- `GET /v1/nodes/<node_ident>/management/subscriptions/subscription_bmc_id`

  Retrieves a sbuscription. Returns a JSON object representing the chosen subscription (`subscription_bmc_id`).

  Error codes:

  - 404 Not Found if node or subscription is not found.

```
{
  "id": "<subscription_bmc_id>",
  "destination": "<destinatination_url>",
  "protocol": "<protocol>",
  "context": "<context>",
  "event_types": ["Alert"]
}
```

- `POST /v1/nodes/<node_ident>/management/subscriptions`

  Requests the creation of a subscription.

  - Required: *destination*.

  HTTP codes:

  - 201 Created
  - 400 Bad Request

```
{
 "destination": "http(s)://host/path",
}
```

- `DELETE /v1/nodes/<node_ident>/management/subscriptions/<subscription__bmc_id>`

  Requests the deletion of a subscription

  HTPP codes:

  - 204 No Content
  - 404 Not Found

Note

The PATCH verb is not being supported at this time in this feature.

Client (CLI) impact

The following commands will be created:

```
baremetal node create subscription [node_uuid] [destination]
baremetal node subscription delete [subscription_uuid]
baremetal node subscription list [node]
baremetal node subscription show [node] [subscription_uuid]
```

openstacksdk

Add support for the event subscriptions in openstacksdk.

RPC API impact

The following new RPC calls will be added:

- Create subscription

```
def create_subscription(self, context, node_id, destination, topic=None):
```

- Delete subscription

```
def delete_subscription(self, context, node_id, subscription_bmc_id,␣
↪topic=None):
```

- List subscriptions

```
def get_all_subscriptions(self, context, node_id, topic=None):
```

- Get a subscription

```
def get_subscription(self, context, node_id, subscription_bmc_id,␣
↪topic=None):
```

Driver API impact

The *ManagementInterface* will be updated with the following functions:

```
def create_subscription(self, task, destination):
    """Add the new subscription object to the BMC."""

def delete_subscription(self, task, subscription_bmc_id):
    """Remove the subscription from the BMC."""

def get_all_subscriptions(self, task):
    """List all subscriptions from the BMC"""

def get_subscriptions(self, task, subscription_bmc_id):
    """Get a subscriptions from the BMC"""
```

The above methods are implemented for Redfish hardware types. We will disallow changing the management interface of a node if there are any subscriptions.

Nova driver impact

None.

Ramdisk impact

None.

Security impact

It is recommended to use https.

Other end user impact

The user wont be able to choose the `EventTypes` for the subscription, since the option is deprecated in Redfish EventDestination v1_5_0. We will be using *Alert* by default for the `EventTypes`.

The user wont be able to choose the `Protocol` for the subscription, by default it will be *Redfish* following the schema for EventDestination.

Scalability impact

None.

Performance Impact

None.

Other deployer impact

None.

Developer impact

Other drivers may implement this feature if the BMC has support for event subscription.

4.2.3 Implementation

Assignee(s)

Primary assignee:
<iurygregory, iurygregory@gmail.com>

Redfish Implementation Details

The actual support for EventDestination in sushy is based on schema[2], since HW vendors are still working on adding support for newer versions where the property `EventTypes` is deprecated. Based on this the Ironic API will only accept the following redfish properties to create a subscription:

- Destination - Required

By default we are considering `Protocol` as *Redfish*, `EventTypes` as *[Alert]* and `Context` as .

When vendors have the support for newer EventDestination new fields will be added to the Ironic API.

[2] https://redfish.dmtf.org/schemas/v1/EventDestination.v1_0_0.json

Work Items

- Add support for Events Subscriptions in sushy[1] Page 174, 2.
- Add event subscription support to ManagementInterface
- Add event subscription support to redfish hardware type
- Add RPC for event subscriptions
- Add REST API for event subscriptions

4.2.4 Dependencies

None.

4.2.5 Testing

- Unit Tests
- Tempest tests

4.2.6 Upgrades and Backwards Compatibility

No upgrade impact.

4.2.7 Documentation Impact

- API reference will be added
- Client documentation will be added.

4.2.8 References

4.3 Add new node name filter API

https://bugs.launchpad.net/ironic/+bug/1526319

This blueprint proposes adding a way to filter nodes in the API by their name (regex,wildcard).

GET /v1/nodes/?name_regex=<regexp_str> GET /v1/nodes/?name_wildcard=<wildcard_str>

4.3.1 Problem description

Current there is only api GET /v1/nodes/<node_name> to exactly retrieve the ironic node via node_name. However for customer, if regular and wildcard expressions filters are supported to retrieve the nodes via user input, that should be useful for user to filter nodes by name flexible, especially for the users who have a lot of baremetal nodes which are managed by ironic.

Possible use case is to support client bulk operations, such as to power off some nodes which are filtered out by node name regexp or wildcard, with single command.

[1] https://redfish.dmtf.org/schemas/v1/EventService.v1_0_8.json

4.3.2 Proposed change

Add new API:

- GET /v1/nodes/?name_regex=<regexp_str>
- GET /v1/nodes/?name_wildcard=<wildcard_str>

We support both BRE and ERE IEEE POSIX Regular expression standard[1].

And will add db api support, for different databases, there are different operators for regular expression:

- postgresql: ~
- mysql: REGEXP
- sqlite: REGEXP

So we will check the current database which ironic is using, and get the regular expression operator from the above map. If there is no such db in the above supporting map, will raise api exception to user and tell user the current database is not supported for this name regex query api.

For wildcard filter, all database can support, because will run with LIKE SQL operator which is standard SQL.

Alternatives

Use node tags for client bulk operations.

Data model impact

None.

State Machine Impact

None.

REST API impact

Add new API GET /v1/nodes/?name_regexp=<regexp_str>, and GET /v1/nodes/?name_wildcard=<wildcard_str>, and bump API micro-version. Please see Proposed change.

Client (CLI) impact

Add new command that executes two new API:

- ironic node-list name-regex=<regexp_str>
- ironic node-list name-wildcard=<wildcard_str>

Have a similar modification to the OSC plugin.

RPC API impact

None.

Driver API impact

None.

Nova driver impact

None.

Ramdisk impact

N/A

Security impact

None

Other end user impact

None.

Scalability impact

None.

Performance Impact

None.

Other deployer impact

None.

Developer impact

None.

4.3.3 Implementation

Assignee(s)

Primary assignee:
whaom

Work Items

- Add new API GET /v1/nodes/?name_regex=<regexp_str>
- Add new API GET /v1/nodes/?name_wildcard=<wildcard_str>
- Add new name_regex filter option for ironic node-list command.
- Add new name_widcard filter option for ironic node-list command.
- Add new name_regex filter option for the existing commands which take node id as input to support bulk operation.
- Add new name_widcard filter option for the existing commands which take node id as input to support bulk operation.

4.3.4 Dependencies

None.

4.3.5 Testing

Will add unit test code to cover the new api.

4.3.6 Upgrades and Backwards Compatibility

None.

4.3.7 Documentation Impact

Update API document adding a new API

4.3.8 References

[1] https://en.wikipedia.org/wiki/Regular_expression#Standards

4.4 Obtaining Steps

https://storyboard.openstack.org/#!/story/1719925

https://storyboard.openstack.org/#!/story/1715419

In Ironic, we have a concept of steps[1] to be executed to achieve a task utilizing a blend of driver code running in the conductor and code operating inside of the ironic-python-agent.

In order for this to be useful, we have to be able to raise the visibility of what is available to be performed to the end user of the API. Presently users are only able to rely upon documentation, and the state of the code including modules that could be loaded in.

This issue is further compounded as the entire list of steps is a union of information identified from the `ironic-conductor` process managing the node and the `ironic-python-agent` process executing upon the node.

Note

This document is present in the backlog as there are implementation issues to this feature. Please see Gerrit change 606199 for more information.

4.4.1 Problem description

- API users presently have to rely upon documentation of steps to know what is available.
- Different steps may be available with different hardware managers.
- With the increasing use of the Deploy Steps[2] framework, new steps should be anticipated to be added with new releases of Ironic.
- The `ironic-python-agent` must be running to obtain a complete list of steps.

[1] Manual cleaning - https://specs.openstack.org/openstack/ironic-specs/specs/5.0/manual-cleaning.html
[2] Deploy Steps - https://specs.openstack.org/openstack/ironic-specs/specs/11.1/deployment-steps-framework.html

4.4.2 Proposed change

In order to keep this solution relatively lightweight, there are four fundamental changes that will be needed in order to facilitate visibility.

This doesnt seek to solve complete visibility by creating additional processes, but instead seeks to provide tools to collect data, with the limiting factor being we can only return the current available information.

How to do it?

Step 1

The initial step is to provide an API endpoint that returns the current available list of steps visible for a node running in the conductor. This would be an API endpoint, to a RPC method, to a conductor manager method, which would then return the list of steps, while tolerating the absence of `ironic-python-agent`.

Note

The ironic community consensus is that this feature should cache steps and return those cached steps as available to the user.

Step 2

Addition of a `hold` provision state verb and `holding` state.

Note

During a specific planning and discussion meeting to determine the path for a feature such as this, the ironic community reached a consensus on the call that a holding state would be useful, and could likely be implemented aside from the API functionality proposed in this backlog specification.

Initial State	*Temporary State*	*Possible next verbs*
manageable	holding	manage, clean, provide, active, inspect
available	holding	active, manage, provide

With the invocation of the state:

- The machine is moved to the provisioning network.

Note

There is a slight issue with this transition in that to clean the node would realistically need to be on the cleaning network. Operationally changing the DHCP address is problematic as we have learned with the rescue feature.

- The deployment ramdisk is booted.
- The `ironic-python-agent` would then be left in a running state, allowed to heartbeat (or be polled), and the API endpoint added in the prior step would fetch a complete list of steps that can be executed upon.

Alternatives

An alternative to this solution would be to provide an async API endpoint to perform the steps detailed in step 2, and cache the data which could then be retrieved by the user asynchronously. In this case, the user would have to poll the API to determine if the cached information has been updated.

The conundrum is that this would have to be constrained by states, which means we would still need to build state machine states around this to represent the current operation to users.

Data model impact

None

State Machine Impact

As noted above, we would add a new hold verb, which would allow transition back to the prior state. This `hold` verb would only be accessible from the `manageable` and `available` states.

In this holding state, API users would be able to request logical next steps, in-line with the present state, as detailed in the table above.

REST API impact

The node object returned would expose additional `provision_state` states, however this is a known quantity with all state machine impacts.

An additional provision state target verb of `hold` to trigger the state machine change.

An endpoint will be added on to enable an API user to return the list of known steps via the RPC interface and the conductor, which will be triggered as a GET request.

Note

Community consensus is that we should not be initiating a synchronous call to IPA to collect data, that we should instead return cached data and somehow trigger the cache to be updated.

Example:

```
GET /v1/nodes/{node_ident}/steps[?type=(clean|deploy)]
{
  [{"source": "conductor",
    "deploy": [
      {
        "interface": "deploy",
        "step": "deploy",
        "priority": 100,
      },
    ],
    "clean": [
      {
        "interface": "deploy",
        "step": "erase_devices",
        "reboot_requested": False,
        "priority": 10,
```

(continues on next page)

(continued from previous page)

```
        "abortable": True,
      },
      {
        "interface": "bios",
        "step": "apply_configuration",
        "args": {....},
        "priority": 0,
      },
      {
        "interface": "raid",
        "step": "create_configuration",
        "args": {....},
        "priority": 0
      },
      {
        "interface": "raid"
        "step": "delete_configuration",
         "args": {....},
         "priority": 0
      }
    ]
  },
  {"source": "agent",
  ...
  }
  ]
}
```

If a specific `type` is requested, then the request shall only return the requested type of steps. If no type is defined, both sets will be returned to the caller.

Normal response code: 200 Expected error codes:

```
* 400 with malformed request
* 503 upon conductor error
```

Note

API micro-version will be incremented in accordance with standard procedure.

Client (CLI) impact

ironic CLI

None

openstack baremetal CLI

An `openstack baremetal node steps` and `openstack baremetal node hold` commands will be added to facilitate returning the data exposed by this api.

RPC API impact

A new RPC method will need to be added called `get_steps` that will support a single argument to indicate what class of steps are being requested by the API user.

Driver API impact

None

Nova driver impact

None is required for this feature.

That being said, there is value to enable a node to be scheduled which is being held for an available deployment. As such, it could be an optional enhancement which could save quite a bit of time in a deployment process. This could be enabled by allowing nova to consider a node in the `holding` state to be available for deployments by also evaluating the `target_provision_state` for nodes in `holding`. It would be fairly tight coupling, but a frequent ask is for faster deployments, and it would be a route that we could take to enable such functionality in terms of holding for deployment.

Ramdisk impact

None

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

None

Developer impact

None

4.4.3 Implementation

Assignee(s)

Primary assignee:
Julia Kreger (TheJulia) <juliaashleykreger@gmail.com>

Other contributors:
?

Work Items

- Implement API to retrieve a list of states.
- Implement State machine changes to allow an idle agent instance to return cleaning step data.
- Add API tests to ironic-tempest-plugin.
- Update state machine documentation.
- Add Admin documentation.
- Update CLI documentation.

4.4.4 Dependencies

None

4.4.5 Testing

Basic API contract and state testing should be sufficient for this feature.

4.4.6 Upgrades and Backwards Compatibility

N/A, The existing rolling upgrades and RPC version pinning practice should be more than sufficient to support this feature.

4.4.7 Documentation Impact

Additional details will need to be added to the Admin guide. State documentation will need to be updated. Update client documentation for new state verb.

4.4.8 References

4.5 Support a new hardware type for Fujitsu PRIMEQUEST MMB

https://storyboard.openstack.org/#!/story/1726271

This spec proposes adding a new hardware type that supports deployment of servers managed by Management Board(MMB) for Fujitsu PRIMEQUEST 3000 Series. MMB is a system control unit that performs management tasks, including control and monitoring in the cabinet, partition management, and system initialization.

4.5.1 Problem description

Since PRIMEQUEST definitely differs from iRMC interface, Ironic cannot handle Fujitsu PRIMEQUEST servers by using `irmc` hardware type at present. Therefore, this spec proposes a new hardware type for MMB in order to handle PRIMEQUEST by ironic. PRIMEQUEST has multiple partitions. Each partition works as a physical server. That is, a partition relates an ironic node. In addition, several partitions can be managed by one MMB. In order to control a specified partition, the ironic node has to know its partition number.

4.5.2 Proposed change

This spec proposes the `fujitsu-mmb` hardware type, implementing the Power, Management and Console. The hardware type uses ssh library in order to connect and execute commands into MMB.

Based on this premises, to be enrolled, the node MUST have the following parameters:

driver_info

- **(Required) mmb_address**
  - IP address of the MMB to ssh into.
- **(Required) mmb_username**
  - Username to authenticate as.
- **(Required) mmb_partition**
  - Partition number to manage.
- **mmb_ssh_key_filename:**
  - Filename of optional private key(s) for authentication. If **mmb_ssh_password** is also specified, it will be used for unlocking the private key. It recommends to store at shared volume like NFS or CIFS.
- **mmb_ssh_password**
  - Password to use for authentication or for unlocking a private key. At least one of this or **mmb_ssh_key_filename** must be specified.
- **mmb_ssh_port**
  - Port on the node to connect to. Default is 22.

Well define a new class:

- `fujitsu_mmb.MMBHardware`

Following interfaces will be implemented:

- `MMBPower`
- `MMBManagement`
- `MMBConsole`

`MMBPower`
Ironic sets/synchronizes this interfaces. After synchronization, this interface controls the power state of the nodes using MMBs command.

`MMBManagement`
: This interface allows the user to get and set the boot-order of a server hardware by executing the command on MMB.

`MMBConsole`
: This interface provides serial console view by executing the command on MMB.

This hardware type will support `PXEBoot` for boot and `ISCSIDeploy`, `AgentDeploy` for deploy.

Alternatives

None

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

ironic CLI

None

openstack baremetal CLI

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

None

Security impact

This hardware type retrieves following information.

- SSH private key filename
- SSH password for MMB

However, ironic only stores a filename of SSH private key into driver_info as **mmb_ssh_key_filename**. SSH Key information doesnt include in REST API body. Regarding SSH password, it will be hidden in REST API body like *******.

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

None

Developer impact

None

4.5.3 Implementation

Assignee(s)

Primary assignee:
y-furukawa-2

Other contributors:
shiina-hironori

Work Items

- Implement new `fujitsu-mmb` hardware type and interfaces.
- Implement unit-test cases for `fujitsu-mmb` hardware type and following interfaces.
  - MMBPower
  - MMBManagement
  - MMBConsole
- Write documents about `fujitsu-mmb` hardware type.

4.5.4 Dependencies

python-pqclient: In order to connect to MMB and execute commands for them.

4.5.5 Testing

During next year, well add 3rd party CI for `fujitsu-mmb` hardware type.

4.5.6 Upgrades and Backwards Compatibility

None

4.5.7 Documentation Impact

`Fujitsu MMB driver` section will be included in Administrators Guide.

4.5.8 References

- Fujitsu PRIMEQUEST http://www.fujitsu.com/us/products/computing/servers/mission-critical/
- MMB http://manuals.ts.fujitsu.com/file/11627/CA92344-0541.pdf
- python-pqclient https://github.com/openstack/python-pqclient

4.6 Support per driver sensor meters

https://blueprints.launchpad.net/ironic/+spec/support-per-driver-sensor-meters

This blueprint is for changing the Ironic Conductor _send_sensor_data interface implementation to support sending sensor meters from providers other than IPMI to Ceilometer.

4.6.1 Problem description

The current implementation of _send_sensor_data is IPMI specific in a number of ways. The Conductor will only acquire sensors from a node which is deployed (has an instance_uuid) and the notifications sent by the Conductor to Ceilometer go to an IPMI specific Ceilometer plugin via event type hardware.ipmi.metrics.

I would like to be able to provide health (and potentially other) sensor information from an iLO Management driver by implementing an iLO specific driver.management.get_sensors_data() interface. The sensor information provided by the iLO Management driver should not be treated as IPMI sensors. Also, health information about a platform should be available even if the node has not yet been deployed with software.

4.6.2 Proposed change

The Ironic Conductor _send_sensor_data routine should always call the underlying driver.management.get_sensors_data routine at each poll interval. It should be the responsibility of the underlying management get_sensors_data routine to determine if it is appropriate to gather sensor information for the node that it manages. If a driver determines it is not appropriate to gather sensor information for a node, the driver should simply return an empty sensor list.

The Conductor should not need to interpret who the sensor provider is in order to send the sensor information along to the metering system.

I see the Conductors role in sensor metering as follows:

- Call Management Driver to read sensors at a specifiable interval
- Create per meter naming using the information provided in the sensors returned by the Management Driver. This will ensure consistent meter naming across multiple providers. It will also enable consistent meter naming for other metering systems (like Monasca) in the future.
- The Conductor must include the Ironic Node UUID in the message sent to the metering system so that the Ironic Node UUID will be included in the sensor meters that are created upon receipt of the message.

With the current implementation, per meter naming is not the responsibility of the Conductor, but the responsibility of the metering system plugin.

I would like to propose a generic Ceilometer sensor notification plugin that will not need to be modified to support new sensor types. To create such a plugin will require that the Ironic Conductor format sensor meters with sufficient information for the metering system plugin to create the samples per sensor meter without performing any sensor data transformations. To accomplish this, the Ironic Conductor will need to implement some operations currently being performed by the Ceilometer plugin, like sensor meter naming, Resource ID naming, and sensor reading parsing. Also, any sensor fields added by the Ceilometer notification plugin, like node will need to be created by the Ironic Conductor.

I would like to introduce a per sensor provider ID field in the driver returned sensor dictionary so that consumers of the sensor resource_metadata can identify the driver that provided the sensor. This field will be named sensor_provider and it will be set to a unique text value per driver. For the ipmitool driver, the value will be ipmi and for the HP iLO driver, it will be ilo. Adding this field to each sensor record will require changing each driver that implements the get_sensors_data interface.

In order to support these Conductor changes, a new Ceilometer notification plugin will be needed and the naming for sensors in Ceilometer will need to be changed. See the Dependencies section for the Ceilometer blueprint associated with creation of the generic sensor Ceilometer plugin.

Alternatives

An additional Ceilometer plugin could be created for each sensor provider, but this would require quite a bit of code duplication in each plugin. Also, the Conductor would need to have a mapping from the Ironic driver producing the sensors to the appropriate Ceilometer plugin that the sensor should be sent to.

The generic sensor Ceilometer notification plugin and associated Conductor changes for supporting multiple providers could be implemented without any changes to the currently defined Ceilometer meter naming. This would be accomplished by placing the sensor provider name in the Ceilometer meter name. i.e. hardware.ilo.temperature for an iLO driver provided temperature sensor.

Data model impact

None

REST API impact

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Security impact

None

Other end user impact

To support provider independent meter naming will require changing the user visible meter names for sensors in Ceilometer. The new provider independent naming will not be backward compatible with the names used for sensor meters in the Juno release. Here is an example of Ceilometer plugin created meter names for sensors in Juno.

```
| Name                    | Resource ID
| hardware.ipmi.current    | IronicNodeUUID-power_meter_(0x16)
| hardware.ipmi.temperature| IronicNodeUUID-16-system_board_(0x15)
```

The proposal is to drop the ipmi component string from the meter name so that meter names become provider independent. Transforming the above to the proposed meter naming would result in the following meter names in Ceilometer.

```
| Name                | Resource ID
| hardware.current    | IronicNodeUUID-power_meter_(0x16)
| hardware.temperature| IronicNodeUUID-16-system_board_(0x15)
```

There are also non Ironic sensor polling agents implemented in Ceilometer which use the ipmi component string in their meter names. These polling agents could updated to adhere to the proposed ironic naming as well. The impact on these polling agents would be to remove the .ipmi identifier from the meter names generated by the polling agent as is shown above. Examples of meter names generated by the IPMI and Intel Node manager polling agents are as follows:

```
| Name                          | Resource ID
| hardware.ipmi.current         | CONF.host-IPMI_SensorID
| hardware.ipmi.temperature     | CONF.host-IPMI_SensorID
| hardware.ipmi.node.temperature| CONF.host
| hardware.ipmi.node.power      | CONF.host
```

Note: Even though hardware.ipmi.node.* meters appear to be IPMI sensor types, they are in fact vendor specific Intel Node Manager sensors.

Modifications to the polling agents to change their sensor naming convention is not within the scope of work defined by this specification.

Scalability impact

None

Performance Impact

None

Other deployer impact

Changing the Conductor to send notification messages to a new generic sensor Ceilometer plugin will require updating Ceilometer in conjunction with the Conductor if Ironic to Ceilometer sensor metering is enabled.

Developer impact

There will be a Conductor dependency on a the generic Ceilometer plugin in order for sensor metering to occur. Developer coordination of changes to Ceilometer and the Ironic Conductor will be necessary for verification of operation.

4.6.3 Implementation

Assignee(s)

Primary assignee:
<jmank@hp.com>

Other contributors:
<None>

Work Items

- Change the Conductor to remove all IPMI sensor related assumptions.
- Change the get_node_info_list query to not filter based on associated being true and move instance_uuid check into the ipmitool and ipminative drivers.
- Create a SensorMetrics base class and CeilometerSensorMetrics derived class to implement meter naming, sensor packaging and sensor posting. The Conductor will be modified to instantiate a CeilometerSensorMetrics class at initialization and invoke the SensorMetrics send_sensors interface from the the Conductor _send_sensor_data periodic timer routine.

The intent of the SensorMetrics class is to encapsulate any sensor data transformations necessary for the targeted metering system.

```
@six.add_metaclass(abc.ABCMeta)
class SensorMetrics(object):
    @abc.abstractmethod
    def send_sensors(self, context, task, sensors):
        """Send Sensors in the list to the metering system

        : param context: request context
        : param task: TaskManager instance with shared lock
        : param instance_uuid: Running instance UUID or None if not
                               deployed
```

(continues on next page)

(continued from previous page)

```
        : param sensors: List of Sensors to send to metering system
        """
```

4.6.4 Dependencies

This work is dependent on the following Ceilometer blueprint and specification.

- https://blueprints.launchpad.net/ceilometer/+spec/generic-notification-sensor-meter-plugin

4.6.5 Testing

Unit tests will be need to be added to verify the new code paths. I plan on doing ProLiant platform testing of the capabilities as well.

4.6.6 Upgrades and Backwards Compatibility

These changes to the Conductor will require installation of the Ceilometer generic sensor notification plugin if the sending of sensor data messages is enable for the Conductor via send_sensors_data=true. If Ironic is updated without adding the generic sensor notification plugin, sensor data messages will be sent, but they will not show up as meters in Ceilometer. Documentation will need to be updated to indicate that Ironic Conductor sensor data sending is dependent on the generic Ceilometer notification plugin for Ceilometer.

If the new meter naming scheme is adopted, prior sensors already in the Ceilometer database will retain their prior naming, so post upgrading to the new Conductor and generic sensor Ceilometer plugin will cause a loss in continuity in sensor naming. Also, any existing Ceilometer sensor meter queries based on the Juno sensor meter naming scheme will need to be changed to use the Kilo sensor meter naming scheme.

4.6.7 Documentation Impact

Documentation changes to docs/source/deploy/install-guide.rst will be necessary to cover the visible functional changes as well as the upgrade dependency with Ceilometer.

4.6.8 References

- [openstack-dev] [Ironic][Ceilometer] Proposed Change to Sensor meter naming in Ceilometer http://lists.openstack.org/pipermail/openstack-dev/2014-October/048631.html

CHAPTER

FIVE

IMPLEMENTED SPECIFICATIONS

These specifications have been implemented and are grouped in the development cycles in which they were completed.

5.1 Xena

5.1.1 18.1

Anaconda deploy interface

https://storyboard.openstack.org/#!/story/2007839

Problem description

We would like Ironic to provide the following features (described in more detail later in this spec):

- Create LVM logical volumes, volume groups, and physical volumes
- Create MD raid arrays
- Create Linux filesystems
- Generate Linux fstab files based on created Linux filesystems

Support for creating LVM and MD structures has significant usefulness in when provisioning Nodes with multiple physical storage devices.

Use-cases include:

- A user wants to use a LVM or MD feature
- A user wants to create a Linux filesystem on any block device created during provisioning
- A user wants an fstab with entries for all filesystems created by the Ironic deploy interface

While Ironic supports creation of custom MD raid arrays using deploy templates, it could lead to an explosion of flavors[1].

Proposed change

This spec suggests implementing these block device provisioning features by creating a new anaconda deploy interface using Anaconda. The anaconda deploy interface allows a user to select one of a set of pre-defined Kickstart configurations[2].

[1] https://specs.openstack.org/openstack/ironic-specs/specs/approved/deploy-templates.html#current-limitations
[2] https://pykickstart.readthedocs.io/en/latest/kickstart-docs.html

The primary advantage of this approach is in allowing customization of the deployed operating system with all features offered by Kickstart commands.

This spec is interested in the following Anaconda features. In most cases, each feature described has a 1:1 relationship with a Kickstart command:

Feature	Ironic	Anaconda
Zeroize GPT/MBR structures	yes	yes
Create a partition in GPT/MBR	yes	yes
Install a bootloader (GRUB)	yes	yes
Create a LVM logical volume	no	yes
Create a LVM volume group	no	yes
Create a LVM physical volume	no	yes
Create a MD raid array	limited	yes
Create filesystems specified by the user	no	yes
fstab entry generator	no	yes

As shown, there are several block device-related features supported by this spec.

In addition, there are some desired pseudo-features, for example LVM physical volume on MD raid array that are accomplished by the user providing sequences of (imperative) Kickstart commands that effectively implement this.

Deploying a node with `Anaconda` deploy driver involves solving following problems:

1. Identifying the correct Anaconda version to use with the OS image.
2. Getting user-provided kickstart file to the Anaconda runtime.
3. The Operator/User generating an OS image with necessary tools in a format supported by Anaconda.
4. Passing correct kernel cmdline arguments to Anaconda by generating them in PXE boot driver.
5. Sending status back to Ironic API when the deployment starts/ends or crashes.
6. Handling cleaning when Anaconda deploy driver is used.

Scope

The scope of the Anaconda deploy interface is limited to

- CentOS/RHEL >= 7 and Fedora >=31
- Anaconda will be used to deploy the OS image.
- Support both UEFI and Legacy BIOS mode.

Matching the OS image with correct anaconda version

Anaconda is a two stage installer where the ramdisk is considered stage1. Once stage1 is loaded, it tries to download stage2 of the installer over the network. Stage2 is a squashfs[3] image and the location of stage2 can be specified using `inst.stage2` kernel command line argument.

```
inst.stage2=http://<address>:<port>/httproot/<node-uuid>/squashfs.img
```

[3] https://en.wikipedia.org/wiki/SquashFS

To deploy the OS using anaconda, apart from the OS image,kernel,ramdisk(stage1) and anaconda squashfs(stage2) images are required. All these artifacts should be uploaded to glance and associated with the OS image by the operator.

```
openstack image create --disk-format raw --container-format compressed \
    --file path/to/os_image.tar.bz2 \
    --property kernel_id=$MY_VMLINUZ_UUID \
    --property ramdisk_id=$MY_INITRD_UUID \
    --property stage2_id=$MY_ANACONDA_SQUASHFS_UUID \
    --property os_distro=RHEL \
    --property os_version=7 centos-7-base \
    --property ks_template=glance://uuid``
```

This is a departure from how direct deploy interface works where the kernel_id and ramdisk_id are either in configuration file or set in driver_info. IPA is distro agnostic, Anaconda is not. There is no single Anaconda installer version that is compatible with all major versions of CentOS. Each major version of CentOS has its own version of anaconda installer. For this reason we require the operator to associate correct PXE kernel, PXE ramdisk and Anaconda squashfs with the OS image as properties in Glance. The Anaconda deploy interface shall validate the image properties and make sure that all required properties are set before the deployment.

Kickstart templates

Anaconda installer needs a kickstart file to deploy an operating system non-interactively. The kickstart file is used to automate the deployment of the operating system. The default kickstart template when not modified by the operator should automatically partition the available disks using the autopart mechanism[4] and deploy the OS. The default kickstart file template will be named default_kickstart.template and referenced by the configuration option `default_ks_template` in ironic.conf under `[kickstart]` section.

Example default kickstart template:

```
lang en_US
keyboard us
timezone America/Los_Angeles --isUtc
#platform x86, AMD64, or Intel EM64T
text
install
cmdline
reboot
selinux --enforcing
firewall --enabled
firstboot --disabled
auth --passalgo=sha512 --useshadow


bootloader --location=mbr --append="rhgb quiet crashkernel=auto"
zerombr
clearpart --all --initlabel
autopart
```

[4] https://pykickstart.readthedocs.io/en/latest/kickstart-docs.html#autopart

All kickstart templates will be automatically appended with following mandatory sections during deployment

```
# Downloading and installing OS image using liveimg section is mandatory
liveimg --url={{ liveimg_url }}

# Following %pre, %onerror and %trackback sections are mandatory
%pre
/usr/bin/curl -X PUT -H 'Content-Type: application/json' -H 'Accept:␣
↪application/json' -d '{.."agent_status": "start"}' http(s)://host:port/v1/
↪heartbeat/{{node_ident}}
%end

%onerror
/usr/bin/curl -X PUT -H 'Content-Type: application/json' -H 'Accept:␣
↪application/json' -d '{"agent_status": "Error: Deploying using anaconda.␣
↪Check console for more information."}' http(s)://host:port/v1/heartbeat/{
↪{node_ident}
%end

%traceback
/usr/bin/curl -X PUT -H 'Content-Type: application/json' -H 'Accept:␣
↪application/json' -d '{.."agent_status": "Error: Anaconda crashed␣
↪unexpectedly."}' http(s)://host:port/v1/heartbeat/{{node_ident}
%end

# Sending callback after the installation is mandatory
%post
/usr/bin/curl -X PUT -H 'Content-Type: application/json' -H 'Accept:␣
↪application/json' -d '{.."agent_status": "success"}' http(s)://host:port/v1/
↪heartbeat/{{node_ident}
%end
```

Multiple %pre, %post, %traceback and %error sections can exist in a kickstart file. These sections will be processed and executed in the order they are encountered[5].

Custom kickstart templates should be uploaded to glance or hosted in a webserver accessible by the conductor or on the conductors filesystem.

The operator can set the kickstart file using URI formats `glance://<uuid>` or `http(s)://host:port/path/ks.cfg` or `file://path/to/ks.cfg`

If the API user decides to store the kickstart file in glance they can do so by running the following command

```
openstack image create --file ks.cfg --container-format bare \
    --disk-format raw custom_kickstart_template
```

Users can specify a specific kickstart template for a node via the nodes instance_info field, with key ks_template. For example:

[5] https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/installation_guide/sect-kickstart-syntax#sect-kickstart-preinstall

```
openstack baremetal node set $NODE_UUID \
    --instance_info ks_template=glance://uuid

or

openstack baremetal node set $NODE_UUID \
    --instance_info ks_template=http(s)://port:host/path/ks.cfg

or

openstack baremetal node set $NODE_UUID \
    --instance_info ks_template=file://path/to/ks.cfg
```

The user can also associate a kickstart template with an OS image(image_source) in glance. The template specified in the instance_info will take precedence followed by `ks_template` property on the OS image. Finally if `ks_template` property is not present in both instance_info and OS image then the default kickstart template specified in the configuration file will be used.

The custom kickstart template will be downloaded and stored in httproot/<node-uuid>/ks.cfg. Where as `httproot` is defined in `http_root` configuration item under `[deploy]` section of `ironic.conf` configuration file. Once the custom kickstart template is downloaded it will be validated against `os_distro` and `os_version`

```
ksvalidator -v RHEL7 ks.cfg
```

os_distro and os_version are properties of image_source(The OS image). The API user is required to set os_distro and os_version. If there is no os_distro or os_version set on the image_source, the kickstart file will be validated against `DEVEL` version of kickstart syntax. See `ksvalidator -l` for list of supported kickstart versions[6].

The OS_DISTRO should be one of RHEL and OS_VERSION should be either 7 or 8.

The kickstart file is passed to anaconda installer using the kernel cmdline argument `inst.ks`

```
inst.ks=http(s)://<address>:<port>/httproot/<node-uuid>/ks.cfg
```

Kernel command line arguments

Two important kernel command line arguments are required for the anaconda installer to work.

1. `inst.stage2`
2. `inst.ks`

Both of these kernel command line arguments will be appended to `pxe_options` dictionary. A function similar to get_volume_pxe_options() will be added to pxe_utils to facilitate this.

OS image format

While Anaconda supports installing individual RPM packages from a remote server, the deployment driver will only support installation of disk image formats described by liveimg[7]. liveimg accepts tarballs, squashfs images and any mountable disk images. Users can generate squashfs images and tarballs.

[6] https://pypi.org/project/pykickstart/
[7] https://pykickstart.readthedocs.io/en/latest/kickstart-docs.html#liveimg

Deployment status

Anaconda installer doesnt know how to talk to Ironic APIs. However we can have %pre and %post sections of kickstart file make API calls to Ironic. The ramdisk will use heartbeat API to talk to the Ironic API. The %pre %onerror, %traceback, and %post sections will be populated with `curl` calls to heartbeat API when conductor renders the kickstart template.

The %pre and %post sections of kickstart files are executed in order they are encountered by anaconda.

At the start of the installation following status will be sent using %pre section of the kickstart file from anaconda ramdisk to lookup

POST {callback_url: , agent_token: <token>, agent_version: ,

agent_status: start}
http(s)://<address>:<port>/v1/heartbeat/{{node_ident}}

On receiving the start status from anaconda ramdisk, the conductor will set driver_internal_info[agent_status] = start

At the end of the OS installation %post section will be used to send following message back to Ironic

POST {callback_url: , agent_token: <token>, agent_version: ,

agent_status: success}
http(s)://<address>:<port>/v1/heartbeat/{{node_ident}}

On receiving the success the conductor will move the Ironic node to active state depending on the current state and set driver_internal_info[agent_status] = success

If there are errors during installation we will capture those error using %onerror[8] and %traceback[9] sections of kickstart file, then send the error status to Ironic

POST {callback_url: , agent_token: <token>, agent_version: ,

agent_status: Error: <msg>}
http(s)://<address>:<port>/v1/heartbeat/{{node_ident}}

On receiving the Error status the conductor will set the provision_state of Ironic node to deploy_failed depending on the current status of the node and set the last_error field of the Ironic node.

There will be no calls to /v1/lookup API from ramdisk. The agent token will be generated when the kickstart file is rendered. Agent token will be embedded in the kickstart file for the heartbeat `curl` call to use. The driver avoids calls to lookup API because it is difficult to read and extract agent token using scripts in the ramdisk.

Cleaning

The `PXEAnacondaDeploy` driver will inherit from `AgentBaseMixin` interface and `DeployInterface` similar to `PXERamdiskDeploy` driver. This implies that the cleaning will be done by the agent Driver not by the Anaconda deploy driver.

During deployment the `PXEAnacondaDeploy` driver will use the properties associated with the `image_source` to figure out the deploy_kernel and deploy_ramdisk. However during cleaning it will use the driver_info[deploy_kernel] and driver_info[deploy_ramdisk] fields to determine the cleaning kernel and ramdisk. This mean the driver_info deploy_* fields should refer to IPA kernel/ramdisk. This change is likely to be a point of confusion.

[8] https://pykickstart.readthedocs.io/en/latest/kickstart-docs.html#chapter-7-handling-errors
[9] https://pykickstart.readthedocs.io/en/latest/kickstart-docs.html#chapter-8-handling-tracebacks

Alternatives

The Anaconda deployment driver is specific to Red Hat based distributions. This deployment driver wont support distributions not supported by Anaconda. For example ubuntu is not supported by this deploy driver. A similar driver can be implemented to support ubuntu using preseed[10] files.

Another alternative is to define a generic partition configuration format and use that configuration instead of kickstart file. This new generic partition configuration will be validated by the conductor and sent to Ironic python agent during deployment. IPA will read the generic partition configuration and use libraries like blivet[11] to partition/format the disks. With this approach the deploy driver isnt tied to a specific distribution or vendor. We explored this approach but we found it to be too complex and reinventing lot of things the distribution installers already do.

The kickstart file is either uploaded to glance or hosted in a webserver in current proposal. Alternatively we can add a new field named `kickstart` to `nodes` table which accepts raw kickstart file

```
openstack baremetal rebuild --kickstart path/to/ks.cfg <node-uuid>
```

Data model impact

None

State Machine Impact

None

REST API impact

Add an optional field `agent_status` to v1/heartbeat API, which can be used to receive deployment status from the anaconda deploy driver.

POST {.. agent_status: <status>} /v1/heartbeat/{{node_ident}}

Client (CLI) impact

None

openstack baremetal CLI

None

openstacksdk

None

RPC API impact

RPC API needs to be updated to handle the new `agent_status` in heartbeat API.

[10] https://wiki.debian.org/DebianInstaller/Preseed
[11] https://pypi.org/project/blivet/

Driver API impact

None

Nova driver impact

There is no impact to novas Ironic driver at this time.

Ramdisk impact

There is no impact to Ironic-python-agent.

Security impact

The `heartbeat` method implemented by the driver has to be unauthenticated so that anaconda can POST to the status API without a token. An attacker could potentially cause targeted denial of service attack by sending invalid/incorrect status to Ironic nodes since the API is unauthenticated. This issue is mitigated by mandatory agent token verification.

Other end user impact

An OS image that can be deployed via liveimg kickstart command should be uploaded to glance along with relevant anaconda installers PXE kernel, ramdisk and squashfs image. The PXE kernel,ramdisk and squashfs need to be associated with the OS image.

```
openstack image set IMG-ID --property kernel_id=$MY_VMLINUZ_UUID \
    --property ramdisk_id=$MY_INITRD_UUID --property \
    squashfs_id=$MY_ANACONDA_SQUASHFS_UUID
```

The end user can make use of their custom kickstart templates during deployment by working with the Operator. The Operator can set the instance_info `ks_template` key with the path of user provided kickstart template. The kickstart template can be in glance `glance://uuid`, webserver `http(s)://host:port/path/ks.cfg` or on the filesystem `file://etc/ironic/ks.cfg` of the conductor.

```
openstack baremetal node set $NODE_UUID --instance_info ks_template=<TMPL>
```

Scalability impact

None

Performance Impact

None

Other deployer impact

The operator has to set default kickstart template under `[kickstart]` section of Ironic configuration file.

```
[kickstart]
default_ks_template=$Ironic_CONF_DIR/kickstart/default_ks.template
```

The `kickstart` deploy interface must be set on the node .. code:: bash

    openstack baremetal node set <NODE> deploy-interface kickstart

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
zer0c00l, sagarun@gmail.com

Work Items

1. Definition of default kickstart template and configuration items related to kickstart deploy template.
2. Implementation of core deploy driver that fetches artifacts from glance, generates PXE configuration files, renders kickstart templates into httproot
3. A CI job to test the anaconda deploy driver
4. Documentation for operators and users

Dependencies

None

Testing

- This driver should be testable in gate. Enhancements might be needed to gate to get this working.
- Devstack support will be added for this driver so that it can be tested easily.

Upgrades and Backwards Compatibility

None

Documentation Impact

Clear operator and user documentation need to be added on kickstart deploy interface and how to make use of it.

References

5.1.2 18.0

Provide Redfish BIOS Attribute Registry data in API

https://storyboard.openstack.org/#!/story/2008571

This proposal is to retrieve the Redfish BIOS Attribute Registry when using the Redfish driver and provide it as additional data in the bios API endpoint.

Problem description

The Redfish BIOS Attribute Registry (aka BIOS Registry) is defined in the DMTF Redfish specification[0] as a way to describe the semantics of each property in a BIOS settings resource. This JSON encoded registry contains descriptions of each property and other information, such as data type, allowable values, and whether the attribute is read-only.

When a user attempts to set a BIOS attribute through clean steps[1] it is not obvious which name should be used for the attribute, what its allowable values are, or even if the attribute is writeable. It is useful to expose the BIOS Registry through an API so that it can be used as a form of documentation when setting BIOS attributes.

An example of an entry in BIOS Registry returned from a BMC for an attribute:

```
{
  "AttributeName": "SriovGlobalEnable",
  "CurrentValue": null,
  "DisplayName": "SR-IOV Global Enable",
  "DisplayOrder": 2331,
  "HelpText": "Enables or disables the BIOS configuration of Single Root
              I/O Virtualization (SR-IOV) devices. Enable this feature
              if booting to a virtualization operating system that
              recognize SR-IOV devices. ",
  "Hidden": false,
  "Immutable": false,
  "MenuPath": "./IntegratedDevicesRef",
  "ReadOnly": false,
  "ResetRequired": true,
  "Type": "Enumeration",
  "Value": [
    {
      "ValueDisplayName": "Enabled",
      "ValueName": "Enabled"
    },
    {
      "ValueDisplayName": "Disabled",
      "ValueName": "Disabled"
    }
  ],
  "WarningText": null,
  "WriteOnly": false
}
```

This change was discussed at the Wallaby mid-cycle, see notes[2].

[0] DMTF Redfish Specification - http://redfish.dmtf.org/schemas/DSP0266_1.11.0.html
[1] BIOS configuration - https://github.com/openstack/ironic-specs/blob/master/specs/approved/generic-bios-config.rst
[2] Discussion at Wallaby mid-cycle - https://etherpad.opendev.org/p/ironic-wallaby-midcycle

Proposed change

The effort will encompass the following changes:

- Support for getting the BIOS Registry in Sushy. Retrieving the BIOS Registry requires multiple Redfish requests as the base URI - `/redfish/v1/Registries`, provides the BIOS Registry URI with a vendor specific name. The registry will be a list of dictionary entries, one for each BIOS attribute.
- Changes to Ironic to retrieve the Redfish BIOS Registry for a node when it gets the BIOS settings; currently this is when Ironic moves the node to `manageable` or `cleaning` and after changing the BIOS settings.
- Filtering of the BIOS registry received from Sushy to only save the entries that match the BIOS settings received from Sushy, based on the `AttributeName` field in the BIOS Registry entries.
- Storing the registry data in the Ironic DB associated with each BIOS setting.
- Exposing the BIOS Registry data via the REST API. No new endpoints will be added. The registry data will be included in the `/v1/nodes/<node>/bios` endpoint only if `?detail=True` is set. The registry data will always be included in `/v1/nodes/<node>/bios/<setting>`. This is similar to how the API for nodes currently works e.g. `/v1/nodes`, `/v1/nodes?detail=True`, and `/v1/nodes/<node>`

Alternatives

- An operator can continue to set the BIOS settings in clean steps perhaps by using vendor documentation or only making simple changes to Boolean values.
- Instead of retrieving the registry and caching it, Ironic could do a synchronous get from the API. However, this may lead to performance problems especially when handling multiple API requests.

Data model impact

- There is a schema defined for the BIOS Registry, for example[3], however it is vendor dependent as to which attribute fields are stored in the registry. Sushy should parse the following common fields which are present in the registry of typical vendors, for example Dell, HPE, and SuperMicro:

```
* AttributeName
* DefaultValue
* Type
* Immutable
* ReadOnly
* ResetRequired
* IsSystemUniqueProperty
* LowerBound
* UpperBound
* MinLength
* MaxLength
* Value (possible values for enumeration types)
  * ValueName
```

[3] AttributeRegistry schema - https://redfish.dmtf.org/schemas/v1/AttributeRegistry.v1_3_4.json

- The BIOS Registry will be stored in the Ironic DB along with the BIOSSetting, see[4]. The following columns will be added and must be included in the migration code:
  - `attribute_type` (string)
  - `allowable_values` (list)
  - `lower_bound` (integer)
  - `max_length` (integer)
  - `min_length` (integer)
  - `read_only` (boolean)
  - `reset_required` (boolean)
  - `unique` (boolean)
  - `upper_bound` (integer)

For each setting only the relevant fields for `attribute_type` will be stored as returned from the BMC. For example, for an Enumeration type only `allowable_values` will be stored and for an Integer only `min_length` and `max_length`.

In addition, the BIOSSetting `name` field should be indexed for faster retrieval.

State Machine Impact

The BIOS Registry will be retrieved from the BMC upon transition to `manageable` and `cleaning` states using multiple Redfish requests. This may add some additional time to the state transition but it is not foreseen that this will have an impact on the state transition performance.

REST API impact

- No new REST API endpoints will be added. The `/v1/nodes/<node>/bios/<attribute>` endpoint will be changed to include the new registry fields. The `/v1/nodes/<node>/bios` endpoint will only include this data per attribute if `?detail=True` is set, similar to how the `/v1/nodes/` endpoint works. The query can also be used to retrieve only particular fields.
- If BIOS Registry could not be retrieved from the node then the registry fields will be set to `null`.
- For allowable_values, some vendors include both ValueName and ValueDisplayName in the response. The API will just show a list of the allowable values.
- An example response for `/v1/nodes/<node>/bios/ProcVirtualization`, an enumeration type, with the new registry fields is as follows:

```
{ "ProcVirtualization":
    { "created_at": "2021-03-31T13:50:51+00:00",
      "updated_at": null,
      "name": "ProcVirtualization",
      "value": "Enabled",
      "allowable_values": ["Enabled","Disabled"]
      "lower_bound": null,
      "max_length": null,
```

(continues on next page)

[4] BIOSSetting in Ironic DB - https://opendev.org/openstack/ironic/src/branch/master/ironic/db/sqlalchemy/models.py#L326

(continued from previous page)


```
      "min_length": null,
      "read_only": false,
      "reset_required": null,
      "type": "Enumeration",
      "unique": null,
      "upper_bound": null,
      "links": [
      {
        "href": "http://IP/v1/nodes/1b1c6edf-4459-4172-b069-5c6ca3e59e03/
↪bios/ProcVirtualization",
        "rel": "self"
      },
      {
        "href": "http://IP/nodes/1b1c6edf-4459-4172-b069-5c6ca3e59e03/
↪bios/ProcVirtualization",
        "rel": "bookmark"
      }
    ]
  }
}
```


- An example response for `/v1/nodes/<node>/bios/SerialNumber`, a String type unique to this node, is as follows:


```
{ "SerialNumber":
    { "created_at": "2021-03-31T13:50:51+00:00",
      "updated_at": null,
      "name": "SerialNumber",
      "value": "Q95102Q8",
      "allowable_values": [],
      "lower_bound": null,
      "max_length": 16,
      "min_length": null,
      "read_only": false,
      "reset_required": null,
      "type": "String",
      "unique": true,
      "upper_bound": null,
      "links": [
      {
        "href": "http://IP/v1/nodes/1b1c6edf-4459-4172-b069-5c6ca3e59e03/
↪bios/SerialNumber",
        "rel": "self"
      },
      {
        "href": "http://IP/nodes/1b1c6edf-4459-4172-b069-5c6ca3e59e03/
↪bios/SerialNumber",
        "rel": "bookmark"
      }
    ]
```


(continues on next page)

(continued from previous page)

```
  }
}
```

Client (CLI) impact

baremetal CLI

- The bios command `baremetal node bios` will change as follows.
  - This command will now include the registry data as an additional fields, for example:

```
$ baremetal node bios setting show hp-910 ProcVirtualization -f json
{
  "created_at": "2021-03-31T13:50:51+00:00",
  "name": "ProcVirtualization",
  "updated_at": null,
  "value": "Enabled"
  "allowable_values": ["Enabled","Disabled"]
  "lower_bound": null,
  "max_length": null,
  "min_length": null,
  "read_only": false,
  "reset_required": null,
  "type": "Enumeration",
  "unique": null,
  "upper_bound": null,
}
```

- A new parameter `--long` will be added to the `baremetal node bios list` command to include the registry data for each attribute. It will not be included by default. This is similar to `baremetal node list <node> --long` command.

openstacksdk

Openstacksdk does not currently have support for bios settings. Although not a requirement for this specification. bios should be added to openstacksdk and the detailed registry information should be included.

RPC API impact

A new method to get the BIOS registry will be added to the RPC API.

Driver API impact

- The `cache_bios_settings` method in the Redfish driver will now also get the BIOS Registry from Sushy.

Nova driver impact

None

Ramdisk impact

None

Security impact

Unprivileged access to the BIOS configuration can expose sensitive BIOS information and configurable BIOS options to attackers, which may lead to disruptive consequence. Its recommended that this kind of ability is restricted to administrative roles.

Other end user impact

None

Scalability impact

The Redfish requests to retrieve BIOS Registry data will increase the traffic to the BMC however, since these requests will only be when the node transitions to `manageable` or `cleaning`, it should not impact scalability.

The BIOS Registry data will be stored in the Ironic DB along with the BIOS setting that it corresponds to. This will add to the size of the Ironic DB but its not expected to be prohibitive as the BIOS settings list, although it varies per vendor, is approximately 100 to 200 items.

Performance Impact

It is not expected that this change will introduce a performance impact on the time it takes for nodes to transition to `manageable` or `cleaning`.

Other deployer impact

None

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
bfournie@redhat.com

Work Items

- Add support for BIOS Registry to Sushy
- Add caching support for the registry to Ironic.

- Add retrieval of the registry in Ironic when transitioning the node to `cleaning` or `manageable` and store/update the cache.
- Add the API to get the BIOS Registry for the node.
- Add configuration items for `bios_registory_lang` and `bios_registry_enable` to ironic.conf.

Dependencies

None

Testing

- Unit testing will be added with sample vendor data but it will necessary for 3rd party testing to be added to test each vendors interface.

Upgrades and Backwards Compatibility

None

Documentation Impact

- API reference will be updated.

References

5.2 Wallaby

5.2.1 17.0

Apply pre-defined system hardware configuration in single step

https://storyboard.openstack.org/#!/story/2003594

Ironics popularity and the number of ironic deployments, both integrated OpenStack and stand-alone, continue to grow. Alongside that, the number of bare metal systems managed by each deployment is increasing. Those trends have led operators to demand that improvements be made to the operational management of hardware system configuration BIOS settings, RAID, boot mode, network ports, inventory, and more. Solutions must reduce operational costs and the time required to fulfill a tenants request for a newly deployed and activated bare metal system.

Use cases which would benefit from this proposal include:

- Correct, consistent, and rapid hardware configuration during deployment of fleets of bare metal systems with similar hardware capabilities
- Production environments in which physical systems are repurposed for workloads that require different configurations and fast turnaround is at a premium
- Inventory of individual system configurations

To achieve that, prospective solutions should offer:

- Creation of bare metal system configurations from common or golden systems
- Management of configurations stored at a designated storage location

- Application of a stored configuration to a bare metal system
- Acquisition of a systems resulting entire configuration following application of a complete or partial configuration
- Acceleration of hardware configuration by reducing the amount of time, often via fewer required reboots, which is dependent on system and ironic hardware type support

This specification proposes a generally applicable solution comprised of new cleaning and deploy steps which process system configurations. Each hardware type may implement it according to its and its systems capabilities.

Problem description

A large number of systems with identical or similar hardware arrives. The user wants all of their hardware configured the same.

- In ironic, that can be a tedious, error-prone task.
- Ironic currently has APIs to configure BIOS and RAID, but it does not offer interfaces for all subsystems for which vendors offer configuration.
- Applying a configuration can take a great deal of time, often requiring multiple reboots.

The same challenges apply when different workflows are run on the same system, each needing a different hardware configuration.

Proposed change

Introduction

The following concept is used in this specification:

configuration mold
: A document which expresses the current or desired configuration of a hardware system. It also offers an inventory of a systems hardware configuration.

This specification proposes to:

- Add new, optional cleaning[1] and deploy[2] steps which, in a single step, can completely configure a systems hardware. They can apply the saved configuration of a system with sufficiently similar hardware capabilities. Steps which obtain and persistently store a systems configuration are also proposed.
  - `export_configuration`
    : obtain a systems hardware configuration and persistently store it as a *configuration mold* so it can be used to configure other, sufficiently capable systems and as an inventory. More details are available in *Export configuration*.
  - `import_configuration`
    : apply the hardware configuration described by an existing, persistently stored *configuration mold* to a sufficiently capable system. More details are available in *Import configuration*.
  - `import_export_configuration`
    : import and then export the hardware configuration of the same system as a single, atomic

[1] https://docs.openstack.org/ironic/latest/admin/cleaning.html#cleaning-steps
[2] https://docs.openstack.org/ironic/latest/admin/node-deployment.html#node-deployment-deploy-steps

step from the perspective of ironics cleaning and deployment mechanisms. This can further reduce the time required to provision a system than a sequence of separate import and export steps. More details are available in *Import and export configuration*.

- Define the data format of the vendor-independent content of a *configuration mold*. It contains a section for each hardware subsystem it supports, RAID and BIOS. The desired configuration of a subsystem for which ironic has been offering configuration support, again RAID and BIOS, is expressed as much as possible in ironic terms.

  To support hardware configuration beyond what ironic defines, but which vendors, systems, and their hardware types could offer in a system-specific manner, a third section, OEM, is proposed. The OEM section can also be used to specify configuration ironic already supports. The content of the OEM section is specific to the system and its hardware type.

  More details, along with an example, are in section *Format of configuration data*.

- Describe the persistent storage of *configuration molds* for both integrated OpenStack and stand-alone ironic environments and how ironic and its users interact with them. See *Storage of configuration data*.

- Declare what is considered a complete implementation for the purposes of this specification. An implementation will be considered complete if it offers all three (3) pairs of cleaning and deploy steps and supports all of the defined *configuration mold* sections, presently BIOS and RAID. Additionally, persistent storage of *configuration molds* must be supported for both ironic environments, integrated OpenStack and stand-alone.

- Offer an alternative, not a replacement, to ironics currently available granular configuration steps which operate on a single subsystem: RAID and BIOS. How it compares to ironics present functionality is described in *Alternatives*.

- Outline a minimum viable product (MVP) implementation by the `idrac` hardware type in section *idrac-redfish implementation*. We aim for it to offer all three (3) pairs of cleaning and deploy steps. In the MVP, only the OEM *configuration mold* section will be supported. Both methods of persistent *configuration mold* storage will be implemented and supported.

- Illustrate possible implementation for generic `redfish` hardware type in section *redfish implementation*.

Goals

With this specification, we are going to achieve the following goals:

- Define an ironic hardware type framework which offers more operationally efficient bare metal hardware configuration

- Facilitate a consistent method to accelerate bare metal hardware configuration, necessitating fewer reboots, when supported by a hardware system and its ironic hardware type

- Describe a first, MVP implementation by the `idrac` hardware type

Non-goals

The following are considered outside the scope of this specification:

- Implementation of the approach by all hardware types; it is optional

- Requiring a hardware types implementation be complete; it may be partial

Export configuration

The export configuration clean/deploy step extracts existing configuration of indicated server (golden server) and stores it in designated storage location to be used in *Import configuration* clean/deploy step.

Clean/deploy step details are:

Interface
Management interface

Name
`export_configuration`

Details
Gets the configuration of the server against which the step is run and stores it in specific format in indicated storage as configured by ironic.

Priority
0

Stoppable
No

Arguments

- URL of location to save the configuration to

Sample of clean/deploy step configuration:

```
{
  "interface": "management",
  "step": "export_configuration",
  "args": {
    "export_configuration_location": "https://server/edge_dell_emc-poweredge_
↪r640"
  }
}
```

The workflow of configuration export consists of 3 parts:

1. Get current nodes configuration (driver specific)
2. Transform the configuration to common format (transformation is driver specific; format is common, see *Format of configuration data*)
3. Save the storage item to designated storage (common to all drivers, see *Storage of configuration data*)

Usage of *export_configuration* is not mandatory. If the configuration is acquired previously or in another way, user can also upload it to storage directly.

Import configuration

Once the configuration is available, user can use it in the import configuration clean/deploy step to configure the servers.

Clean/deploy step details are:

Interface
Management interface

Name
`import_configuration`

Details
Gets pre-created configuration from storage by given location URL and imports that into given server.

Priority
0

Stoppable
No

Arguments

- URL of location to fetch desired configuration from

Sample:

```
{
  "interface": "management",
  "step": "import_configuration",
  "args": {
    "import_configuration_location": "https://server/edge_dell_emc-poweredge_
↪r640"
  }
}
```

The workflow of the import configuration consists of 3 parts:

1. Using given configuration location and ironics storage settings, get the configuration from the storage (common to all drivers).
2. Transform the configuration to driver specific format (driver specific)
3. Apply the configuration (driver specific)

- Sections that are not specified in the *configuration mold* are left intact, for example, it is possible to configure only subset of BIOS settings and other BIOS settings and RAID settings remain unchanged.
- If an error is encountered, the clean/deploy step fails. On failure, no assurances can be made about the state of the systems configuration, because the application of the configuration is system and ironic hardware type dependent and there are many possible failure modes. A defined subsystem configuration sequence and transactional rollback semantics do not seem to apply.
- When a step fails, the ironic node is placed in the `clean failed` or `deploy failed` state and the nodes `last_error` field may contain further information about the cause of the failure.
- Successful application of configuration specified has no side effects on the nodes fields (like BIOS and RAID configuration).

Warning

Depending on each vendors capabilities importing can be powerful step that allows configuring various things. Users and vendors need to be aware of these capabilities and make sure not to overwrite settings that are not intended to be replaced, for example, deleting RAID settings or static BMC IP address.

Import and export configuration

Import and export configuration clean/deploy step is composite step that executes both importing and exporting one after another as atomic operation. This can be used to get the inventory just after configuration and can be useful when not all aspects of system are being configured, but need to know the outcome for all aspects.

Clean/deploy step details are:

Interface
Management interface

Name
`import_export_configuration`

Details
Gets pre-created configuration from storage, imports that into given server and exports resulting configuration.

Priority
0

Stoppable
No

Arguments

- URL of location to fetch desired configuration from
- URL of location to save the configuration to

Sample of clean/deploy step configuration:

```
{
  "interface": "management",
  "step": "import_export_configuration",
  "args": {
      "import_configuration_location": "https://server/edge_dell_emc-
↪poweredge_r640"
      "export_configuration_location": "https://server/edge_dell_emc-
↪poweredge_r640_server005"
  }
}
```

The workflow of configuration import and export consists of parts:

1. Execute workflow as in step *Import configuration*
2. When importing succeeds, execute workflow as in step *Export configuration*

Format of configuration data

The format to store the reusable configuration is in JSON format and consists of 3 sections:

- `bios reset` to indicate if reset is necessary before applying settings indicated in the list of BIOS attribute key-value pairs inside `settings` section as in Apply BIOS configuration step[3]. If `reset` is false, then settings that are not included in `settings` sections are left unchanged.
- `raid` as in RAID create configuration step with key-value pair settings and `target_raid_config` property[4]
- `oem` driver specific section with everything else that does not fit into bios and raid sections together with interface name that can handle this data. The interface name can be used to distinguish for which hardware type this configuration data is meant and used for validation during import before trying to parse this section and catch incompatibility early. The data format of this section is controlled by implementing interface and only restriction is that it needs to fit in JSON property.
- There is no overlapping with `oem` and vendor-independent sections, like `bios` and `raid`.
- If overlapping is determined during import, then configuration data is considered invalid and cleaning/deployment step fails.

Sample of exported data format:

```
{
  "bios": {
    "reset": false,
    "settings": [
      {
        "name": "ProcVirtualization",
        "value": "Enabled"
      },
      {
        "name": "MemTest",
        "value": "Disabled"
      }
    ]
  }
  "raid": {
    "create_nonroot_volumes": true,
    "create_root_volume": true,
    "delete_existing": false,
    "target_raid_config": {
      "logical_disks": [
        {
          "size_gb": 50,
          "raid_level": "1+0",
          "controller": "RAID.Integrated.1-1",
          "volume_name": "root_volume",
          "is_root_volume": true,
          "physical_disks": [
            "Disk.Bay.0:Encl.Int.0-1:RAID.Integrated.1-1",
```

(continues on next page)

[3] https://docs.openstack.org/ironic/latest/admin/bios.html#apply-bios-configuration
[4] https://opendev.org/openstack/ironic/src/branch/master/ironic/drivers/raid_config_schema.json

(continued from previous page)

```
            "Disk.Bay.1:Encl.Int.0-1:RAID.Integrated.1-1"
          ]
        },
        {
          "size_gb": 100,
          "raid_level": "5",
          "controller": "RAID.Integrated.1-1",
          "volume_name": "data_volume",
          "physical_disks": [
            "Disk.Bay.2:Encl.Int.0-1:RAID.Integrated.1-1",
            "Disk.Bay.3:Encl.Int.0-1:RAID.Integrated.1-1",
            "Disk.Bay.4:Encl.Int.0-1:RAID.Integrated.1-1"
          ]
        }
      ]
    }
  }
  "oem": {
    "interface": "idrac-redfish",
    "data": {
      "SystemConfiguration": {
        "Model": "PowerEdge R640",
        "ServiceTag": "8CY9Z99",
        "TimeStamp": "Fri Jun 26 08:43:15 2020",
        "Components": [
          {
            [...]
            "FQDD": "NIC.Slot.1-1-1",
            "Attributes": [
              {
              "Name": "BlnkLeds",
              "Value": "15",
              "Set On Import": "True",
              "Comment": "Read and Write"
              },
              {
              "Name": "VirtMacAddr",
              "Value": "00:00:00:00:00:00",
              "Set On Import": "False",
              "Comment": "Read and Write"
              },
              {
              "Name": "VirtualizationMode",
              "Value": "NONE",
              "Set On Import": "True",
              "Comment": "Read and Write"
              },
            [...]
            ]
```

(continues on next page)

(continued from previous page)

```
            }
          ]
        }
    }
}
```

`oem` section of sample data depicts snippets from Dell SCP file (see more at *idrac-redfish implementation*) that has some metadata about the source of the configuration (`Model`, `ServiceTag`, `TimeStamp`) and inside `Components` section there are attributes listed that can be applied during import and is controlled by `Set On Import` property.

Storage of configuration data

Common functionality among hardware types is the configuration storage and will be implemented for all vendors to be used in their implementations.

The requirements for storage are:

1. Support node multi-tenancy
2. Support multiple conductors
3. Support ironic in stand-alone mode
4. Support ironic operators with ephemeral ironic database
5. User can manage (list, create, view, edit, delete) *configuration molds*

To fulfill these requirements a storage solution is proposed:

- Use full URL to indicate *configuration mold* location
- Swift is used as storage backend
  - Full URL points to Swift object within containers
  - Access is restricted by projects/tenants/accounts
  - HTTP GET and HTTP PUT used to get and store data
  - Ironic service account has access to used containers
  - User can manage *configuration molds* by accessing Swift container directly
- Web server is used as storage backend for ironic in stand-alone mode
  - Used only for Ironic stand-alone or when there is only one tenant as this does not guard against accessing other tenant data
  - HTTP GET and HTTP PUT used to get and store data. Web servers need to have HTTP PUT configured (it is not enabled by default)
  - Basic auth used with pre-defined credentials from ironic configuration to access data
  - User can manage *configuration molds* by accessing the web server directly
- In future additional storage back-ends can be added

To implement this, these changes will be made:

New settings added:

```
[molds]storage
[molds]user
[molds]password
```

`[molds]storage` used to define what storage backend used. By default it will be Swift with option to configure Web server. In future more options can be added.

`[molds]user` and `[molds]password` is used when Web server is configured as storage backend. Ironic will use this to encode it in Base64 and add as header to HTTP requests. By default they will be empty indicating that no authorization used.

The workflow for getting stored configuration data:

1. Given *configuration mold*s full URL and used storage mechanism configured in `[molds]storage` fetch data using appropriate credentials.
2. Handle any errors, including access permission errors. If errors encountered, a step fails.

The workflow for storing the configuration data:

1. Given *configuration mold*s full URL and used storage mechanism configured in `[molds]storage` store data using appropriate credentials.
2. Handle any errors, including access permission errors. If errors encountered, a step fails.

Swift support

For Swift as end-user that initiates cleaning or deploying is different from the service user that actually does cleaning or deploying, it is necessary to allow ironic conductor (service user) access Swift containers used in steps. There is security risk having access to all tenant containers that is described in *Security impact* section.

Web server support

For web server authorization Basic authentication will be used from `[molds]user` and `[molds]password`. It is strongly advised to have TLS configured on the web server.

idrac-redfish implementation

For iDRAC to implement these proposed steps it will use Server Configuration Profile (SCP)[5] that allows to export existing configuration server and import the same configuration file to another server. Settings for different sub-systems such as BIOS, RAID, NIC are included in the configuration file.

The implementation would transform configuration between SCP data format and ironic data format. In the first version (MVP), all SCP data is exported to and imported from `oem` section as-is without any transformation. In the following versions this will be improved to start using `bios` and `raid` sections. The implementation will use Redfish protocol. As this is part of OEM section in Redfish service, the communication will be implemented in sushy-oem-idrac library. In next versions after MVP is done, transformation between SCP data format and ironic data format will be implemented in ironic part of idrac-redfish interface.

[5] https://downloads.dell.com/manuals/all-products/esuprt_solutions_int/esuprt_solutions_int_solutions_resources/dell-management-solution-resources_white-papers15_en-us.pdf

When comparing configuration runtime using separate BIOS and RAID configuration jobs versus SCP approach on R640 the difference was 11 minutes versus 7 minutes where SCP was faster within one reboot.

redfish implementation

This section describes how these steps could be implemented for generic `redfish` driver. Compared to the proposed implementation of `idrac-redfish` described in *idrac-redfish implementation* that implements these steps using iDRAC specific functionality, Redfish specification, in contrast, does not have a resource and action that accepts configuration data all together and implementation for `redfish` needs to take different approach.

This illustrates how dedicated resources as they are available in Redfish service can be assembled to be used in single step. It is not part of this spec to implement this.

The following describes how this implementation would support configuration of RAID, BIOS.

For `import_configuration`:

1. Given *configuration mold* that contains `bios` and `raid` sections.
2. Apply RAID configuration to create a volume with immediate application or apply on reset (if changes need reboot).
3. Apply BIOS updates to pending settings with apply time on reset.
4. Reboot the system for pending changes to take effect.

The difference from existing functionality in ironic:

1. There is only 1 step instead of dedicated 2 steps.
2. There are less reboots (depending on necessity to reboot for RAID) as all configuration is assembled before reboot.

For `export_configuration`:

1. Use BIOS resource to get current BIOS settings.
2. Use Storage resource and related to get current RAID settings.
3. Transform results into *configuration mold*s format.

The difference from existing functionality in ironic:

1. There is no step that fetches current configuration across various subsystems.
2. Closest to achieve this would be getting nodes `raid_config` field and getting BIOS attributes and transforming it to deploy template that uses existing RAID and BIOS steps.

For `import_export_configuration` combine implementations of `import_configuration` and `export_configuration` together.

Alternatives

We can continue to support only the current, granular hardware provisioning deploy and clean steps.

The closest currently available functionality in ironic is deploy templates that enable assembling several existing steps together. In the same manner these deploy templates can be re-used for as many systems as necessary. However, comparing deploy templates to the proposed solution currently:

- No functionality to get the configuration from already configured system, user has to construct the initial configuration file themselves by hand or a script. To make it easier, can use cached BIOS and RAID settings from a node that was deployed, but this reuse is still not built in ironic.
- Depending on vendors capabilities each step may require reboot to finish. For example, iDRAC BIOS configuration apply needs reboot to take effect and deem the step to be finished. For now ironic cannot line up several steps that require reboot and then finish them all by one reboot. For next step to start the previous one needs to be finished. The proposal makes it possible to handle this internally inside the import step, that is, if that is how a hardware type is implementing this, it can create 2 tasks for BIOS, RAID configuration and then reboot and watch for both tasks to finish to deem the step as finished.
- Using OEM section each vendor can add support for configuring more settings that are not currently possible using common (vendor-independent) ironic clean/deploy steps.

This proposal does not suggest to replace current clean/deploy steps and deploy templates but add alternative approach for system configuration.

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

None

Security impact

JSON will be used as user input. It will be validated, sanitized, and treated as text. Common storage utils will use Pythons *json.loads* when retrieving and *json.dumps* when storing data. If there is additional validation and clean up necessary for vendor specific implementation, for example, OEM section, then that needs to be added to drivers implementation.

Configuration molds are considered a sensitive data and they also can contain plain text password, for example, for BMC. Implementation for storage of *configuration molds* will support authorization and separation of tenants. Operators will be suggested to add additional security e.g., configure storage back-end encryption and TLS for web server.

As cleaning/deploying is executed by ironic service account and not user that initiated clean or deploy, ironic service account needs access to used Swift containers provided by users. In multi-tenant, end-user accessible Ironic API this could lead to accessing data not belonging to the tenant by, e.g., guessing or somehow finding out another tenants URL and feeding that to ironic that has access to it while end-user does not. This issue needs to be addressed separately in future releases. There are possibly other use cases that are affected by this limitation and would benefit from addressing this.

Other end user impact

The configuration items can accumulate in the storage as there is no default timeout or logic that deletes them after a while because these configuration items should be available after nodes cleaning or deployment. If user do not need the reusable configuration items anymore, then user should delete those themselves from the storage.

This adds new configuration section `[molds]` to control storage location. Default values are provided.

Scalability impact

None

Performance Impact

Depending on hardware type implementation, deployments can become faster. When *configuration mold* is processed, it is read in memory, but it is not expected that these *configuration molds* will be large.

Also based on vendors implementation these can be synchronous or asynchronous steps. If steps are synchronous this will consume a long-lived thread where operators may need to adjust the number of workers.

Other deployer impact

Need to configure storage backend, Swift or web server.

Developer impact

There will be new clean and deploy steps available that each driver can implement. They are optional and other developers can implement those at their own time if needed.

Implementation

Assignee(s)

Primary assignees:

- Aija Jaunteva (@ajya, aija.jaunteva@dell.com)
- Richard Pioso (@rpioso, richard.pioso@dell.com)

Other contributors:
None

Work Items

For common functionality:

- Implement common functionality for configuration storage
  - Swift support
  - Web server support

For `idrac-redfish` implementation:

- Implement initial idrac hardware type derivations of the new clean and deployment steps which use the Redfish protocol (MVP)
- Update the iDRAC driver documentation
- Enhance the idrac hardware type implementation to support the `bios` section of the configuration data
- Enhance the idrac hardware type implementation to support the `raid` section of the configuration data

Dependencies

None

Testing

For now, tempest tests are out of scope, but in future 3rd party continuous integration (CI) tests can be added for each driver which implements the new clean and deploy steps.

Upgrades and Backwards Compatibility

This change is designed to be backwards compatible. The new clean and deploy steps are optional. When an attempt to use them with a hardware type which does not implement them, then clean or deploy will fail with error saying that node does not support these steps.

Documentation Impact

Node cleaning documentation[6] is updated to describe new clean steps under Management interface for `idrac-redfish`.

[6] https://docs.openstack.org/ironic/latest/admin/cleaning.html

References

System Scoped Role Based Access Control

Specification Scope: OpenStack Integrated

https://storyboard.openstack.org/#!/story/2008425

Ironic has long been considered an admin-only service. This changed with recent work contributed to help make Ironic able to provide multi-tenant capabilities. This work was in support of the the efforts of the Mass Open Cloud community to support delineation of access and resource allocation between the participating organizations through an `owner` and `lessee` field.

However there is a growing desire to delineate scopes in which user accounts have access to the API. This effort is sometimes referred to as Secure RBAC in the OpenStack community, which is an initiative to have scope restricted authentication across OpenStack services, where the scoping and modeling is consistent to provide a consistent authorization experience. This is achieved via system scoped role assignments. In this model, an `admin`, `member`, and `reader` exists. The `admin` role implies `member`, and `member` implies `reader`. These roles exist in one of three scopes, `system`, `domain`, and `project`. The relevant scopes for most services in OpenStack are the `system` and `project` scopes.

In essence this effort is to group the access and actions behind personas, which are role and scope permutations that can be applied to a user via role assignments in keystone.users and then ensuring that the invoked access rights do not permit inappropriate access such as edit fields as a reader only role on the system scope. At a high level, this is conceptually modeled into `admin`, `member`, and `reader` roles. During the policy in code, effort the `admin` role was modeled the `baremetal_admin` role and the `reader` role via the `baremetal_observer` role, however none of this is system scoped. The existing access roles are project scoped to a `baremetal` project, and Ironic has no concept of token scopes.

Role definitions:

- **admin - This is in essence an administrative user with-in the operating**
  scope. These are accounts which Create/Delete $things, and in keystone default configuration, this role implies the `member` role. In an Ironic context, we can think of this user as the infrastructure administrator who is adding their baremetal machines into Ironic.
- **member - This is a user which can act upon things. They may be able to read**
  and write with-in objects, but cannot create/delete new objects unless it is an explicitly permitted action. An Ironic example may be that we might want to permit members to be able to request allocations, or change a nodes provision state. Similar to `admin` implying `member`, `member` implies `reader`.
- **reader - This is a user which needs to be able to have read-only access.**
  They can read objects but not change, modify, or delete objects. In a `system` scope it may be a network operations center employee who has a business need to be able to observe the status and details. In a `project` scope, this may be someone attempting to account for resources, or accounts for automated processes/reporting.

Note

Additional details on default role definitions is covered in the Keystone specification define default roles or the Keystone administrators guide.

Note

A future potential is that an `auditor` role may exist, but it would *not* match readers. Auditors would be read-only in nature, but their role would likely allow sensitive values to be unmasked. This has not been decided upon, and depending on service configuration could likely be implemented manually with a custom policy file. That being said, this is out of scope of this specification document at this time. It is also important to note that the `admin`, `member`, and `reader` roles do not automatically unmask sensitive data, and should not be anticipated to do so by default.

When considering the above role definitions and the use case of baremetal, scope differences will have a major difference between what exists today and what would exist moving forward. At a high level, you wouldnt allow a `project` scoped `admin` to have full administrative access to Ironic.

Scope definitions:

- system - This is similar to the existing scope of the cloud deployment today.
- **domain - This scope, is presently only used in Keystone for associations.**
  We do not anticipate this to apply, and the primitives do not exist in Ironic.
- **project - This is the logical grouping in which users are members of projects**
  and have some level of implied member rights with-in that project. This is tracked and associated using the `project_id` value.

Additional information can be found in the *Keystone administration - tokens <https://docs.openstack.org/keystone/latest/admin/tokens-overview.html#authorization-scopes>* documentation and the Keystone contributor - services documentation.

Problem description

The fundamental issue at hand is Ironic does not understand scoped access. This lack of understanding of scoped access coupled with existing project scoped role/access creates a confusing and different authentication experience from other services which have implemented the ability to have scope delineated access, being `Keystone` and `Nova` as of the point in which this specification was authored.

Coincidentally there is a desire from larger OpenStack operators to have the ability to delineate access. In other words permit operations centers to be able to view status, but not be able to act upon nodes. This may be a `reader` scoped user in the scope of a project, or in the scope of the entire system.

As projects within OpenStack implement Scope and Role delineation, and enable scope based access restriction, a risk exists that Ironic will become incompatible with the models attempting to be represented.

And thus we must implement support to delineate scopes, roles, and ultimately what may be a differing access model for some remote resources. In particular, risk exists with existing integrations as they may grow to expect only Project scoped requests, and refuse a System scoped member request. These sorts of issues will need to be identified and appropriately navigated.

In summary, the purpose of this specification is to make changes in *ironic* and *ironic-inspector* to be consistent and future compatible with the rest of the OpenStack community. This will further enable infrastructure operators where they can leverage the prior community policy work in the OpenStack community to override the policy defaults the community reaches.

Proposed change

At a high level, the desire is to:

a) Have greater consistency through the adoption of standard roles.

b) Implement the ability to move to the standard scope based restriction where the new standardized roles would apply.

c) Move services, such as ironic from the concept of *admin projects* to a *system scope*.

We will do this by:

1) Constructing a new set of policies to reflect the secure RBAC model where the scope is included as part of the definition.

2) Deprecating the previous policies in code which consist of roles scoped to the `baremetal` project. These should be anticipated to be removed at a later point in time.

3) Implementing explicit testing to ensure scopes are handled as we expect.

4) Creating an integration test job leveraging the `oslo.policy` setting to enforce scope restriction to help ensure cross-service compatibility and potentially having to alter some cross-service interactions to ensure requests are appropriately modeled. It should be expected that this may make visible any number of possible issues which will need to be addressed.

During the deprecation period, operators will continue to be able to leverage the previous authentication model.

These new policies would model our existing use and data model however with scope applied *and* multi-tenant access enabled. This will enable a friendly default usage path which will still be opt-in unless the node `owner` or `lessee` field is populated on a node object.

Combining the three defined roles of `admin`, `member`, and `reader`, with the three scopes, `system`, `domain`, `project` results in a matrix of possibilities. But, the `domain` is not anticipated to be needed, thus leaving six access scenarios or personas that have to be considered.

Please consult the *High level matrix* for a high level overview as to the anticipated use model.

In order to have a consistent use pattern moving forward, the existing role definitions of `baremetal_admin` and `baremetal_reader` will be deprecated and removed, however they will also not be effective once the `[oslo_policy]enforce_scope` and `[oslo_policy]enforce_new_defaults` parameters are set to `True`.

Above and beyond new policy definitions, the creation of additional tests will be needed in the `ironic` and `ironic-inspector` projects to validate enforcement or appropriate resource denial based upon the scope.

Additional issues and rights validation logic may need to be applied, however that will likely require adjacent/integrated projects to change their policy enforcement.

Note

Adjacent/integrated projects/services, for example is the interaction between Nova, Neutron, Cinder, Glance, Swift, and Ironic. Services do convey context on behalf of the original requester for a period of time, and can make access control decisions based up on this. Ironic has previously had to address these sorts of issues in the Neutron and Cinder integrations.

In terms of `ironic-inspector` and its API, the resulting default policies for this effort would be entirely system scoped and no other scope is anticipated to need implementation as the `ironic-inspector` is purely an admin-only and hardware data collection oriented service.

Note

In review of this specification document, it has been highlighted that a tenant may find it useful to have the ability to trigger inspection of a node, and have it report to *their* own `ironic-inspector` instance. This is an intriguing possibility, but would be a distinct feature above and beyond the scope of this specific work. The benefit of the previous policy in code effort, is operators should be able to simply update the policy in this case, if operationally permissible in that Operators security posture.

High level matrix

The table below utilizes two definitions which hail back to the existing multitenancy work that is present in ironic. They are not the proposed new name, but used to provide conceptual understanding of what the alignment of the policy rule represents since there are technically several different access matrices based upon the variation and ultimately the agreement reached within the community. The end name definition may be something similar, but that is an implementation naming decision, not higher level design decision.

- ***is_node_owner* - When the API consumers project ID value is populated in** the Ironic node objects `owner` field. This represents that they are the authoritative owner of the baremetal node.
- ***is_node_lessee* - When the API consumers project ID value is populated in** the Ironic node objects `lessee` field. This is considered the current or assigned user of the node. See the Allow Leasable Nodes specification for additional details.

Note

It is important to stress, that the table below are general guidelines. A higher level of detail is available below in *Project Scope* and *Endpoint Access Rights*.

Role	System Scope	Project Scope
admin	Effectively the same as the existing baremetal_admin role.	Project `admin` able to have equivalent access to the API as `system` scoped `member` with a filtered view matching *is_node_owner*. `owner` field updates are blocked. Some sensitive fields may be redacted or be restricted from update.
member	New concept for a *do-er* user or service account. Cant add or delete nodes, but can do things like provision_state changes.	Project members will be able to use a baremetal node if *is_node_lessee* or *is_node_owner* is matched and perform field/state updates on individual nodes with the exception of the `owner` and `lessee` fields. Some additional fields or update restrictions will exist.
reade	Effectively the same as the existing baremetal_observer	This is a read-only user concept where a project `reader` would be able to view a node if *is_node_owner* or *is_node_lessee* applies. This role is expected to still have a restricted view, which will likely vary based on which type of granted rights.

Note

An `auditor` role has not been proposed in this work, but *does* make eventual sense in the long term, and should be logically considered as reader does not equal an auditor in role. The concept for `auditor` would expect to allow secrets such as masked fields to be unmasked.

Note

Some role/scope combinations may be combined in discussions and communication in a {scope}-{role} format. This is effectively the persona being defined. Such as *system-admin* for a system wide scope or *project-admin* for a user who is a project administrator.

Note

Field restriction are likely to be controlled by additional policy rules, which MAY cascade in structure where if full general update access is not granted then lower level policies should be enumerated through. Similar logic is already present in ironic.

In effect, a `PROJECT_ADMIN`, if defined in the terms of a rule, would match upon a `project_id` matching the `owner` and the user having an admin role. A `PROJECT_MEMBER` includes `PROJECT_ADMIN` *or* where `project_id` matches `lessee` and the role is `member`.

Alternatives

No alternative is available as the model of implementation. This is due to it attempting to conform to the overall OpenStack model. Fine details should likely be discussed with-in the implementation.

Data model impact

None

State Machine Impact

None

REST API impact

The overall high level behavior of this change will be settings enforced through `oslo_policy` until the deprecated policies are removed from Ironic.

In accordance with API standards, even though it will not modify functional behavior this change will increment the API micro-version. This is to enable API consumers to be able to navigate around possible logic or policy changes around an upgrade. This is unrelated to policy enforcement specifics which cannot be permitted to be visible via the API surface.

End API user behavior is not anticipated to be changed, however with scope enforcement set in `oslo.policy`, an appropriately scoped user will be required.

System Scope

The transition for System scoped roles is fairly straight forward as described by the chart *High Level Matrix* in *Proposed Change*. Existing Admin/Observer roles would be translated to System-Admin and System-Reader respectively.

The addition to this scope is the `member` role concept. This is a user who can *Read* and *Update*, but that cannot *Create* or *Delete* records. In other words, the API consumer can deploy a node, they can update a node, but they are unable to remove a node. They should be able to attach/detach VIFs, and ultimately this should be able to be the rights granted to the service account used by the `nova-compute` process.

A user with a system scope of any valid role type should be anticipated as having full API surface visibility with exception of the special purpose `/v1/lookup` and `/v1/heartbeat` endpoints. This will be different for *Project Scope* based access where nodes will only be visible if owner or lessee are populated.

Todo

Follow-up with neutron regarding port attach/detach.

Todo

Follow-up with Cinder regarding volume attach/detach.

Todo

Follow-up with Nova regarding rights passed through on context.

Note

The primary focus of this specification is targeted at the Wallaby development cycle where the System scope is most beneficial to support. Given time constraints and cross-project mechanics we will likely see additional work to refine scope interactions under this spec as time progresses. Some of these things may be related to `volume` or `port` attachments, or possibly even tighter integration of this functionality in `nova-compute`. All of these things will evolve over time, and we cannot answer them until we reach that point in time.

Project Scope

The Project scoped restrictions in the secure RBAC model are dramatically different, however precedent already exists with the addition of the *is_node_owner* and *is_node_lessee* logic which would apply to project scoped interactions.

API consumers seeking to `GET` resources in the project scope would only be able to view resources which match the `is_node_owner` and/or `is_node_lessee` which are associated to the `owner` and `lessee` fields.

Note

A node CAN have an `owner` and/or `lessee` independently, and at present the policy model delineates access separately.

In this case, a Project-Admin would have similar rights to a System-Member where they would be able to update hardware focused fields such as `driver_info`, however only if `is_node_owner` matches. Project admins who match `is_node_lessee` should not be permitted the ability to update fields such as `driver_info`.

Todo

We may wish to evaluate if it is useful to permit updating `driver_info` as a project admin. Dtantsur thinks, and I agree that this is likely highly deployment and operationly specific, and it may be we need a knob to govern this behavior.

A Project-Member would again be scoped to the appropriate database entries which apply to their users scope. They should be enabled to update fields such as `instance_info`, and provision, unprovision, and potentially update VIFs.

VIFs being set will need to have some additional code to perform an access rights verification to ensure that a project member is attempting to bind to a VIF which matches their node ownership and their users entry, or the value of the lessee field and that requesting users project.

With the physical nature of assets, project scoped users are unable to create or delete any records.

Project scoped readers, again would only have a limited field view with the associated `is_node_lessee` or `is_node_owner`.

Endpoint Access Rights

This list is based upon the published information in the Baremetal API Reference. Not all actions on the node object are covered in this list. Some field restrictions apply. See *Node object field restrictions* for details with a Node object.

Note

This list does not include all possible actions on a node at this time.

Endpoint	Project Scope Accessible
/	Yes, Public endpoint
/v1	Yes, Public endpoint
/v1/nodes	Filtered View and access rights which will necessitate additional policy rules to be added.
/v1/nodes/{uuid}	Filtered view and access rights
/v1/nodes/{uuid}/vendor_pa	No, Will not be permitted as this is a open-ended vendor mechanism interface.
/v1/nodes/{uuid}/traits	Yes, accessible to `owner` to manage
/v1/nodes/{uuid}/vifs	Yes, write access requires additional validations.
/v1/portgroups	Yes, Filtered view and Read-Only for `owner` managability.
/v1/nodes/{uuid}/portgroups	Filtered view and Read-Only
/v1/ports	Yes, Filtered view and access rights for `owner` managability.
/v1/nodes/{uuid}/ports	Filtered view and access rights.
/v1/volume/connectors	Yes, Filtered view, Read-only.
/v1/volume/target	Filtered view, will require extra to prevent target requested is valid for the user/project to request.
/v1/nodes/{uuid}/volume/co	Filtered view, read-only.
/v1/nodes/{uuid}/volume/tar	Filtered view, read-only.
/v1/drivers	No, *system* scope only.
/v1/nodes/{uuid}/bios	Yes, Filtered view based on access rights to the underlying node.
/v1/conductors	No, *system* scope only.
/v1/allocations	Project scoped, however the access model is geared towards owners using this endpoint.
/v1/deploy_templates	No, *system* scope only at this time. as the table/data structure is not modeled for compatibility.
/v1/chassis	No, *system* scope only.
/v1/lookup	No, Agent reserved endpoint.
/v1/heartbeat	No, Agent reserved endpoint.

Warning

Port support will require removal of legacy neutron port attachment through `port.extra['vif_port_id']`

Note

Contributor consensus is that `port` objects do not require project scoped access, however one important item to stress is that the `owner` may be viewed as the ultimate `manager` of a physical node, and the `system`, or `ironic` itself just provides the management infrastructure. This is a valid case and thus it may be reasonable that we settle on permitting owner far more access rights than node lesses in a project scope.

Note

Contributor consensus is that resource class and trait records may only be necessary for a `system` scoped user to edit, however the case can also be made that this should be able to be delegated to the `owner`. This specification, itself, is not calling for a specific pattern, but more so anticipates this will be an implementation detail that will need to be sorted out. It may start as something only `system` scoped users with the appropriate role can edit, and may evolve, or it may not be needed.

Node object field restrictions

Note

These are proposed, however not final. Implementation of functionality will determine the final field behavior and access.

- uuid - Read-Only
- name - Read/Write for Project Admins if the project owns the physical machine.
- power_state - Read-Only
- target_power_state - Read-Only
- provision_state - Read-Only
- target_provision_state - Read-Only
- maintenance - Read/Write
- maintenance_reason - Read/Write
- fault - Read/Write
- last_error - ??? .. TODO:: The issue with `last_error` is that it can leak infrastructure hostnames of conductors, bmcs, etc. For BMaaS, it might make sense?
- reservation - Returned as a True/False for project users.
- driver - Read-Only
- driver_info - Likely returns as an empty dictionary, although alternatively we can strip the URLs out, but that seems a little more complicated.
- driver_internal_info - Likely will return an empty dictionary as Project Admins and Project Members should not really need to see the inner working details of the driver.
- properties - Read-Only
- instance_info - Project Admin/Project Member Read-Write
- instance_uuid - Read/Write for Project Admin/Project Member
- chassis_uuid - Returns None
- extra - Project Admin/Project Member Read-Write .. TODO:: another reason to remove old vif handling logic is the extra field.
- console_enabled - Project Admin/Project Member Read/Write
- raid_config - Read-Only
- target_raid_config - Read-Only

- clean_step - Read-Only
- deploy_step - Read-Only
- links - Read-Only
- ports - Read-Only
- portgroups - Read-Only
- resource_class - Read-Only
- boot_interface - Read-Only
- console_interface - Read-Only
- deploy_interface - Read-Only
- inspect_interface - Read-Only
- management_interface - Read-Only
- network_interface - Read-Only
- power_interface - Read-Only
- raid_interface - Read-Only
- rescue_interface - Read-Only
- storage_interface - Read-Only
- traits - Read-Only
- vendor_interface - Read-Only
- conductor_group - Returns None/Read-only
- protected - Read/Write
- protected_reason - Read/Write
- owner - Read-Only and lessee will be able to see the owner ID.
- lessee - Project Admin/Project Member Read-Write. Lessee will be forbidden from changing the field value.
- description - Read-Write
- conductor - Returns None as it provides insight into the running infrastructure configuration and state, i.e. System visible is the only appropriate state.
- allocation_uuid - Read Only

Special areas:

volume - This represents volume targets and connectors. All values
visible through this path should be read-only. Connector logic should be read/write accessible to Project Admins or Project members where applicable, however additional logic checks need to exist under the hood to validate permission access for the project and user.

state - This is the entry path towards changing state, indicators,
provisioning, etc. This should be permitted for Project Admin or Project Member IF it maps the associated owner or lessee field.

vendor_passthru - Vendor passthrough will not be available to project
scoped users in the RBAC model.

Note

All fields that are scrubbed, i.e. set to None or { } are expected to be read-only fields to project scoped accounts in the new RBAC model.

Client (CLI) impact

openstack baremetal CLI

None anticipated.

openstacksdk

None anticipated.

RPC API impact

At this time, no impact to the RPC API is anticipated. That being said the possibility does exist, given the nature of the security changes, some changes may be required should an additional argument be required. Existing patterns already exist for this and any such changes would be navigated with the existing rpc version maximum and pin capability.

Driver API impact

None.

Nova driver impact

We may wish to go ahead and establish the ability for nova to store the users project ID in the node `lessee` field. In the new use model, this would allow a more natural use pattern and allow users to be able to leverage aspects like power operations or reboot or possibly even rebuild of their deployed instances.

Todo

We should discuss this further. It likely just ought to be a knob for nova-compute with the Ironic virt driver.

Ramdisk impact

None anticipated as the existing heartbeat and lookup resources of the API would not be modified.

Security impact

The overall goal of the Secure RBAC work is to enable and allow an operator to be able to run a service in a more restrictive and constrained model where greater delineation exists between roles.

In a sense, the system scoped operating mode will eventually become the normal operating mode. This is in order to encourage more secure environments, however this will entirely depend upon the default

policies *and* the policies operators put in place which may override the default policy. The overall goal of this specification also being to help identify the new policy mechanics.

In order to help manage this and ensure the overall behavior is enforced as expected, we anticipate we will need to create API behavior testing to ensure operational security and validate that future code changes do not adversely impact permission enforcement.

Other end user impact

No direct end-user impact is anticipated.

Scalability impact

None.

Performance Impact

No direct performance impact is anticipated. The object model already pushes the list filtering down to the DBAPI level, which is ideal for overall performance handling. It is likely some additional checks will produce a slight overhead, but overall it should be minimal and confined to logic in the API services.

Other deployer impact

Cloud infrastructure operators are anticipated to possibly need to adjust `oslo_policy` settings to enable or disable these new policies. This may include cloud operators continuing to use older or other more restrictive policies to improve operational security.

Developer impact

None anticipated at this time.

Implementation

Assignee(s)

Primary assignee:
Julia Kreger (TheJulia) <juliaashleykreger@gmail.com>

Other contributors:
Steve Baker (stevebaker) <sbaker@redhat.com>

Work Items

- Creation of positive/negative policy check tests that represent the current interaction models.
- Creation of scoped policy definitions and associated positive/negative behavior tests:
  - Creation/migration of such for System-Admin where the admin tests appropriately enforce that continity is the same for a scoped admin as with previous tests.
  - Creation/migration of such for System-Reader where values are visible but not able to be written to.
  - Creation of similar for System-Member
  - Creation of similar for Project-Admin

  - Creation of similar for Project-Member
  - Creation of similar for Project-Reader
- Implementation of a CI job which operates a full integration sequence *with* scope policy enforcement enabled via the `[oslo_policy]` configuration.
- Documentation!

Phases

The initial phase for deployment is scoped for the eqiuvalent of the existing project admin scoped authentication for system scoped use.

The next phase, presumably spanning a major release would then cover the project scoped access rights and changes.

Dependencies

Minimum versions of `oslo_policy` will need to be updated to match the Victoria development cycles work product, however this is anticipated to be completed as part of the JSON to YAML policy migration effort.

Testing

An CI integration job is anticipated and should be created or one already leveraged which is utilising the widest configuration of integrated components to ensure that policies are enforced and this enforcement works across components. Due to the nature and scope of this effort, it may be that Ironic alone is first setup to scope limit authorizations as other projects also work in this direction.

Upgrades and Backwards Compatibility

Not applicable.

Documentation Impact

Release note will need to be published with the prior policy deprecation as well as primary documentation updated to reflect the scope based configuration. An in-line documentation warning will likely be necessary depending on what the larger community decides in terms of the RBAC policy efforts and end-user/operator needs to be.

References

- https://review.opendev.org/c/openstack/ironic/+/763255
- https://review.opendev.org/q/topic:%2522secure-rbac%2522+(status:open+OR+status:merged)+project:openstack/ironic
- http://lists.openstack.org/pipermail/openstack-discuss/2020-November/018800.html

5.2.2 16.1

Ironic L3 based deployment

https://storyboard.openstack.org/#!/story/1749193

Ironic relies on L2 network for IP configuration (and optionally PXE booting) during provisioning baremetal nodes. This imposes limitations on Multi-rack deployments and remote site deployments like Edge Cloud etc. Besides that, lossy provisioning network may slow down or fail some deployments.

This specification proposes relying upon Virtual Media boot capability of modern BMCs to deliver boot image, static network configuration, node identification information and more to the node of choice reliably and securely.

Proposed way of booting the node should fully eliminate the need for DHCP and PXE thus enabling deployment over purely L3 network.

Problem description

It is not always easy to provide Layer-2 connectivity between Ironic conductor node and target nodes.

Example use cases:

- Multi Rack deployment: Inter-Rack L2 switching beyond ToR is a challenge with DHCP-relay, as this adds requirement of either having ToR with DHCP-relay, or having a system or VM listening on the remote subnet to relay or proxy DHCP requests.
- Remote site targets: It would be a great idea to deploy Servers in different remote sites from a central location. With the current setup of L2 dependency we need to provide VPN L2 tunnel between Ironic Conductor and remote target server.

  Example: Edge cloud servers or Distributed VNF Flexi Zone Controllers[1].
- Lossy, overloaded provisioning network may cause DHCP or TFTP packet drop slowing down or failing the entire node deployment.

Most of the hardware offerings on the market support booting off virtual media device. While Ironic can boot the nodes over virtual media, its present workflow still relies on DHCP for booted OS to obtain IP stack configuration. This dependency on DHCP can be eliminated if proposed change is implemented.

Proposed change

Deploying a node without DHCP involves solving three crucial problems:

1. Securely delivering boot image to the right node
2. Gathering node configuration information from cloud or site-specific infrastructure
3. Provisioning node configuration (as well as authentication, identification etc) information to the ramdisk operating system running on the node

Virtual media capability of contemporary BMCs coupled with virtual media boot support implemented in some ironic hardware types (e.g. *redfish-virtual-media*) fully solves problem (1). The rest of this spec is dedicated to pondering items (2) and (3).

[1] https://networks.nokia.com/products/flexi-zone

Gathering node configuration

Typical ironic node deployment workflow involves booting Ironic Python Agent (known as IPA or ramdisk) has to be booted to prepare the node. Once set up by IPA, the tenant (AKA instance) operating system is brought up.

In the context of OpenStack, existing cloud infrastructure is capable to manage tenant network configuration (e.g. Neutron), to deliver network configuration to the tenant (e.g. config-drive[2]) and even simplify the application of node configuration (e.g. cloud-init[3], os-net-config[4] or Ignition[11]). All these capabilities together largely solve problems (2) and (3) for the tenants.

However, our software infrastructure does not presently offer much in part of configuring ironic ramdisk networking without DHCP. This spec proposes:

- Allowing the operator to manually associate static node network configuration with ironic node object. The assumption is that the operator would use some other mechanism for IP address management.
- Enable ironic to leverage Neutron for building static network configuration for the node being deployed out of Neutron information.

Provisioning node configuration

This spec proposes burning the contents of *config-drive* containing Nova metadata into the ISO image the node has been booted from in provisioning and state. If no *config-drive* information is supplied to Ironic by the operator, ironic will create one.

To facilitate network configuration processing and application, this spec proposes reusing *network-data.json*[8] Nova metadata format, for Ironic-managed node network configuration.

Example *network-data.json* file:

```
{
    "links": [
        {
            "id": "interface0",
            "type": "phy",
            "ethernet_mac_address": "a0:36:9f:2c:e8:80",
            "mtu": 1500
        },
        {
            "id": "interface1",
            "type": "phy",
            "ethernet_mac_address": "a0:36:9f:2c:e8:81",
            "mtu": 1500
        }
    ],
    "networks": [
        {
```

(continues on next page)

[2] https://docs.openstack.org/nova/latest/user/metadata.html#config-drives
[3] https://cloudinit.readthedocs.io/en/latest/
[4] https://github.com/openstack/os-net-config
[11] https://github.com/coreos/ignition/blob/master/doc/configuration-v3_0.md
[8] https://specs.openstack.org/openstack/nova-specs/specs/liberty/implemented/metadata-service-network-info.html

(continued from previous page)

```
            "id": "provisioning IPv4",
            "type": "ipv4",
            "link": "interface0",
            "ip_address": "10.184.0.244",
            "netmask": "255.255.240.0",
            "routes": [
                {
                    "network": "10.0.0.0",
                    "netmask": "255.0.0.0",
                    "gateway": "11.0.0.1"
                },
                {
                    "network": "0.0.0.0",
                    "netmask": "0.0.0.0",
                    "gateway": "23.253.157.1"
                }
            ],
            "network_id": "da5bb487-5193-4a65-a3df-4a0055a8c0d7"
        },
        {
            "id": "provisioning IPv6",
            "type": "ipv6",
            "link": "interface1",
            "ip_address": "2001:cdba::3257:9652/24",
            "routes": [
                {
                    "network": "::",
                    "netmask": "::",
                    "gateway": "fd00::1"
                },
                {
                    "network": "::",
                    "netmask": "ffff:ffff:ffff::",
                    "gateway": "fd00::1:1"
                },
            ],
            "network_id": "da5bb487-5193-4a65-a3df-4a0055a8c0d8"
        }
    ],
    "services": [
        {
            "type": "dns",
            "address": "10.0.0.1"
        }
    ]
}
```

This spec anticipates associating the content of *network_data.json* document with ironic node object by introducing a new *network_data* field to the node object to contain *network_data.json* information for ironic ramdisk booting.

On the ramdisk side, this spec proposes using Glean[12] for consuming and applying network configuration to the operating system running ironic ramdisk. The main consideration here is that, unlike *cloud-init*, *Glean* is lean, and readily supports a subset of *cloud-init* features.

Alternative ramdisk implementations can choose other ways of bootstrapping OS networking based on *network_data.json* information.

To summarize - in the area of provisioning node network configuration this spec proposes:

- Reusing Nova metadata representation for provisioning network configuration via ramdisk image.
- Adding a new field to ironic node object: *network_data* to use for ramdisk bootstrapping.

  The contents of this field should be validated by ironic conductor API against *Glean* JSON schema and some ad-hoc checks the implementers deem reasonable.

  Having *Glean* schema effectively hardwired into ironic conductor API will not allow for an easy extension or addition of other network configuration formats.
- Creating a new *config-drive* to have it including *network-data.json* file.
- Writing the contents of *config-drive* image into the root of the ISO file system (along with ramdisk and kernel blobs), then making a bootable ISO image.
- Including *Glean* dependency to ramdisk image for managed OS bootstrapping.

However, Ironic rescue operation, at least in its current implementation, will only work if user and provisioning networks are the same network.

Thats because rescue ramdisk will try to renumber NICs of ramdisk by restarting DHCP client. Working around this limitation is out of scope of this spec.

Deployment workflow

To make it easier to grasp, lets reiterate on the entire Ironic deploy work flow focusing on how network configuration is built and used. We will consider two scenarios - stand-alone ironic and ironic within OpenStack cloud. In each scenario we will only consider deploy ramdisk and omit instance booting phases.

Stand-alone ironic

1. Operator supplies deploy ramdisk network configuration, in form of *network-data.json* contents, via *network_data* field of ironic node being deployed. The contents of *network_data.json* must comply to the JSON schema of *network_data.json* that *Glean* can consume.
2. Ironic conductor validates supplied *network-data.json* against *Glean* schema (that is supplied to ironic API program via configuration) and fails early if validation fails. *Glean* schema will not allow any properties of *network_data.json* that cant be applied to the OS by *Glean* even if these properties are valid as Nova metadata.
3. Ironic builds a new *config-drive* image and places *network-data.json* file, with contents taken from *network_data* field, at a conventional location within *config-drive* image.
4. To boot deploy ramdisk, ironic builds bootable ISO out of *deploy_kernel* and *deploy_ramdisk* also writing *config-drive* contents into the root of boot ISO image.

   Glean running inside ramdisk will try to mount virtual CD drive(s), in search for a filesystem labeled *config-2*, read *network_data.json* and apply network configuration to the OS.

[12] https://docs.openstack.org/infra/glean/

Ironic within OpenStack

1. Prior to booting ramdisk, unless operator-supplied network configuration already exists in *network_data* ironic node field, ironic gathers network configuration for each ironic port/portgroup, associated with the node being deployed, by talking with Neutron. Then ironic builds network configuration for ramdisk operating system in form of a *network-data.json* file.
2. Ironic builds a new *config-drive* image and places *network-data.json* file, as build at step (1), at a pre-defined location within *config-drive* image.
3. To boot deploy ramdisk, ironic builds bootable ISO out of *deploy_kernel* and *deploy_ramdisk* also writing *config-drive* contents into the root of boot ISO image.

   Glean running inside ramdisk will try to mount virtual CD drive(s), in search for a filesystem labeled *config-2*, read *network_data.json* and apply network configuration to the ramdisk operating system.

Alternatives

Alternatively to associating the entire and consistent *network_data.json* JSON document with ironic node object, *network_data.json* can be tied to ironic port object. However, experimental implementation revealed certain difficulties stemming from port-centric design, so consensus has been reached to bind *network_data.json* to ironic node object.

Alternatively to make ironic gathering and building *network-data.json*[Page 236, 8], ironic could just directly request Nova metadata service[10] to produce necessary file by instance ID. However, this will not work for non-deploy operations (such as node cleaning) because Nova is not involved.

Alternatively to relying on Nova metadata and *Glean* as its consumer in ramdisk, we could leverage *os-net-config*s feature of taking its compressed configuration from kernel parameters. On the flip side, the size of kernel cmdline is severely limited (256+ bytes). Also, *os-net-config* feels like a TripleO-specific tool in comparison with *cloud-init*, which, besides being a mainstream way of bootstrapping instances in the cloud, understands OpenStack network configuration metadata.

Alternatively to having operator supplying ramdisk network configuration as a *network_data.json* file, Ironic can also accept the entire *config-drive*. That would make it looking similar to instance booting (e.g. Ironic provision state API) and would allow for passing more files to ramdisk in the future.

Alternatively to wiring *Glean* schema validation into Ironic conductor, operator can be asked to validate their *network_data.json* data with some external tool prior to submitting it to Ironic. This would relax ironic conductor dependency on *Glean* input requirements changes and ease *network_data.json* reuse for bootstrapping both ramdisk and instance.

Data model impact

Add a new, user manageable, field *network_data* to ironic node object conveying ramdisk network configuration information to ironic. If set, the contents of this new field should be a well-formed *network-data.json* document describing network configuration of the node running ramdisk.

[10] https://docs.openstack.org/nova/latest/user/metadata-service.html

State Machine Impact

None.

REST API impact

- Update `GET /v1/nodes/detail`, `GET /v1/nodes/{node_id}`, `PATCH /v1/nodes/{node_id}` to add new request/response fields
- `network_data` JSON object conveying network configuration.

Client (CLI) impact

ironic CLI

None

openstack baremetal CLI

- Update `openstack baremetal node create` and `openstack baremetal node set` commands to accept new argument `--network-config <JSON>` with help text describing JSON structure of the network configuration.
- Extend the output of the `openstack baremetal node show` command with `network_data` column.

RPC API impact

None

Driver API impact

- Extend ironic base NetworkInterface with *get_node_network_config* API call providing network configuration for the node being managed. Ironic will burn the output of this API call to the boot image served over virtual media.
- Implement *get_node_network_config* network interface call for non-Neutron networks providing *network_data.json* from *network_data* field of ironic object (if present). The operator could then implement their own IPAM (e.g. for stand-alone ironic use-case).
- Implement *get_node_network_config* network interface call for Neutron networks generating *network_data.json* based on Neutron port and subnet information[9].
- Make virtual media boot interfaces in ironic generating config-drive with *network_data.json* in it if *network_data.json* is not already passed to ironic with the config-drive.
- Make virtual media boot interfaces in ironic writing config-drive contents into root of bootable ISO image it generates on every node boot. Filesystem on this bootable ISO should be labeled *config-2* if it contains config-drive files.

[9] https://github.com/openstack/nova/blob/master/nova/virt/netutils.py#L60

Nova driver impact

None

Ramdisk impact

- Diskimage-builder tool should install *Glean* into ramdisk and invoke on boot.

  On top of that, the *dhcp-all-interfaces* DIB element might not be used anymore because *Glean* will run DHCP on all not explicitly configured (via *config-drive*) interfaces[13].

Security impact

None.

Other end user impact

None.

Scalability impact

None.

Performance Impact

None.

Other deployer impact

None.

Developer impact

None.

Implementation

Assignee(s)

Primary assignee:
etingof (etingof@gmail.com)

Other contributors:
None.

Work Items

- Document *Glean* requirements, with regards to *network_data.json*, in form of machine-readable JSON schema.
- Update ironic node model to include optional, user-specified *network_data* fields carrying ramdisk network configuration in form of *network_data.json* JSON document.

[13] https://opendev.org/opendev/glean/src/branch/master/glean/cmd.py#L323

- Update REST API endpoints to support new *network_data* field
- Support new *network_data* fields in baremetal CLI (*openstack baremetal node*)
- Extend ironic base NetworkInterface with the *get_node_network_config* API call providing network configuration for the node being managed.
- Implement *get_node_network_config* network interface call for non-Neutron networks providing *network_data.json* from *network_data* field of ironic node object (if present).
- Implement *get_node_network_config* network interface call for Neutron networks generating *network_data.json* based on Neutron port and subnet information[Page 240, 9].
- Make virtual media boot interfaces in ironic generating *config-drive* with *network_data.json* on it.
- Make virtual media boot interfaces in ironic writing config-drive files into the root file system of the bootable ISO image it generates on every node boot. The file system should be labeled as *config-2* for *Glean* to find and use it.
- Update ramdisk bootstrapping code to invoke *Glean* on system boot making use of *network-data.json* file if present on the *config-drive*.
- Update diskimage-builder tool to control the inclusion and the options of the new static network configuration processing features.
- Create documentation on DHCP-less boot setup and work flow.

Dependencies

Ramdisk will start depending on *Glean* for processing *network-data.json* document.

Testing

Tempest test of the ironic node deployment using operator-supplied and Neutron-originated *network_data.json*.

Upgrades and Backwards Compatibility

None.

Documentation Impact

Use of L3 based deployment should be documented as part of this item.

References

5.3 Victoria

5.3.1 16.0

In-band Deploy Steps

https://storyboard.openstack.org/#!/story/2006963

Since the Rocky release, Ironic has had support for deploy steps. These allow drivers to customise the node deployment flow. Currently these steps are limited to out of band execution - steps may not be

executed on the node via IPA during deployment. This spec proposes to support execution of in-band deploy steps, in a similar manner to in-band cleaning steps.

Example use cases:

- Software RAID configuration during deployment
- In-band BIOS configuration during deployment

Problem description

Ironics deployment process has historically been quite rigid, not allowing for much customisation outside of the scope of the supported features. This is changing, starting with deploy steps, drivers are able to define custom tasks to perform during deployment using the pattern established for cleaning. The deploy templates feature exposes these deploy steps to users of nova via traits applied to flavors or images.

There are some limitations to deploy steps that we aim to address in this spec:

- deploy steps cannot be executed in-band (on the node, via IPA) during deployment
- deploy steps may only be executed before or after what we will refer to here as the mega step

As we will see, these two issues are linked, since we must break apart the mega step in order to support in-band deploy steps.

Mega step

The mega step starts with the node powered off and configured to boot on the provisioning network. When it has finished, the image has been written to disk, the boot device configured, and the node is plugged into the tenant network and powered on. This covers a significant portion of the deployment process.

Use cases

The following are some use cases for in-band deploy steps.

- as a user of ironic, I want a machine configured with a particular software RAID layout during deployment
- as a user of ironic, I want a machine configured with particular BIOS settings during deployment
- as a user of ironic, I want custom firmware installed during deployment
- as a user of ironic, I want to apply custom NIC configuration during deployment

And here are some for decomposing the mega step.

- as an ironic driver maintainer, I want to create a deploy step that executes with the node booted on the provisioning network
- as an ironic driver maintainer, I want to create a deploy step that executes after the image has been written to disk, while the node is attached to the provisioning network and powered on

Proposed change

Mega step context

The mega step is actually a deploy step called `deploy` on the `deploy` interface. In order to understand how it will be changed, we must first understand what it does and how it fits into the overall deployment

flow. Some details in the following will depend on the particular drivers and configuration in use, but we will try to cover the most common usage.

We start our story in the ironic conductor, where a `do_node_deploy` RPC has been received.

1. `do_node_deploy` RPC received:
    1. validation of `power` and `deploy` interfaces, traits, and deploy templates
    2. the nodes provision state is set to `deploying` (or `rebuilding` if `rebuild` is `True`)
2. execution continues in a new worker thread:
    1. config drive is built and stored, if using one
    2. the `prepare` method of the `deploy` interface is called
    3. deploy steps for execution are determined based on drivers and deploy templates, then stored in `driver_internal_info.deploy_steps`
    4. trigger execution of the next deploy step
3. for each deploy step:
    1. update `driver_internal_info.deploy_step_index` to the index of the current step
    2. execute the deploy step
        1. if it returns `DEPLOYWAIT`, this is an asynchronous step. Exit the worker thread and wait for a `continue_node_deploy` RPC
        2. if it returns `None`, continue to execute the next deploy step
4. if all deploy steps complete:
    1. clean up `driver_internal_info`
    2. start the nodes console, if necessary
    3. the nodes provision state is set to `active`

Well that doesnt look too bad until we realise theres some complexity and asynchronism hidden in there.

Prepare

First of all, lets look at the `prepare` method of the `deploy` interface. For the `agent` and `iscsi` deploy interfaces, this involves:

1. power off the node
2. remove tenant networks (for rebuild, when writing an image)
3. add provisioning network (when writing an image)
4. attach volumes
5. prepare ramdisk boot

So we know that if an image is to be written to the node, then after `prepare`, the node will be ready to boot into the IPA ramdisk on the provisioning network.

Continue node deploy RPC

The next item to unpack here is asynchronous steps and the `continue_node_deploy` RPC. While waiting for execution of an asynchronous step, the nodes provision state is set to `wait callback`. Completion of the step is checked for either in the driver via a periodic task (the driver sets `driver_internal_info.deployment_polling`), or in the heartbeat handler for IPA. On completion, the `continue_node_deploy` RPC is triggered, and the node returns to the `deploying` provision state.

Mega step

The final piece here is the mega step itself: the `deploy` method on the `deploy` interface.

1. reboot the node, wait for heartbeat (when writing an image)
2. on the first IPA heartbeat:
    1. write image to disk synchronously (`iscsi` deploy interface)
    2. call IPA `standby.prepare_image` API, wait for heartbeats until complete (`direct` deploy interface)
3. prepare instance to boot
4. power off the node
5. remove provisioning network
6. configure tenant networks
7. power on the node

At this point, any deploy steps with a lower priority than the mega steps (100) will be executed. Care is required at this point however, since the node is already attached to the tenant network and booting.

In-band cleaning steps

Here is a quick overview of how in-band cleaning works for reference.

In-band cleaning starts by configuring the node on the provisioning network and booting up the IPA ramdisk. On the first heartbeat, the list of in-band steps is queried, combined with out-of-band steps and stored in `driver_internal_info.clean_steps`.

In-band clean steps are advertised by the `deploy` interface, which overrides the `execute_clean_step` method to execute them via the IPA API. In-band clean steps are always asynchronous, with polling triggered via the IPA heartbeat.

In-band clean steps may request that the node is rebooted after completion of the step via a `reboot_requested` flag in their step definition. There is some handling of the case where an IPA returns with a different version after reboot. In this case automated cleaning is restarted, and manual cleaning is aborted.

One final detail is the ability to define hooks that execute after completion of an in-band cleaning step via the `@post_clean_step_hook` decorator.

Observations

In order to gather a list of in-band deploy steps, we need to be able to communicate with IPA. This is first possible at step 2 of the mega step.

We may wish to still allow execution of out-of-band deploy steps before IPA has booted, to avoid unnecessary delays. An example here is BIOS or RAID configuration on Dell iDRACs, which can require the node to be powered on for the lifecycle controller to perform the configuration jobs. An additional boot cycle adds significant delay to the deployment process. This would represent a divergence in behaviour between deployment and cleaning.

When booting from a volume, there may be no image to write. Currently in that case, IPA is not used. This would prevent the use of in-band deploy steps, but booting up IPA just to gather a list of deploy steps would increase the time required to boot from a volume.

The above description of deployment did not cover fast track deploys. This feature allows a node that is already booted with an IPA ramdisk, e.g. from discovery, to bypass the reboot in the mega step.

Proposed mega step decomposition

The following describes the proposed decomposition of the mega step into separate steps.

1. `deploy` [100]:
    1. reboot the node, wait for heartbeat (when writing an image?)
    2. gather in-band deploy steps from the agent
2. `write_image` [80]:
    1. write image to disk synchronously (`iscsi` deploy interface)
    2. in-band deploy step that does the equivalent of the `standby.prepare_image` IPA API and waits for completion of the write (`direct` deploy interface)
3. `prepare_instance_boot` [60]:
    1. install bootloader (if needed)
    2. configure the boot interface
4. `tear_down_agent` [40]:
    1. power off the node
5. `switch_to_tenant_network` [30]:
    1. remove provisioning network
    2. add tenant networks
6. `boot_instance` [20]:
    1. power on the node

The useful priority ranges for inserting custom in-band steps are:

- 99 to 81: preparation before writing the image (e.g. software RAID).
- 79 to 61: modifications to the image before a bootloader is written (e.g. GRUB defaults changes).
- 59 to 41: modifications to the final instance (e.g. software configuration).

deploy

Priority: 100

This deploy step will be largely unchanged from the current `deploy` step. Changes will be necessary for fast track deploys, to skip the direct call to `continue_deploy` and rely on the new deploy steps. For boot from volume this step currently performs the tenant network configuration, instance preparation and reboot, however that can also be moved to the new steps.

write_image

Priority: 80

For the `iscsi` interface, the `continue_deploy` method will be split into a `write_image` deploy step and the `prepare_instance_boot`, `tear_down_agent`, `switch_to_tenant_network`, and `boot_instance` deploy steps.

For the `direct` interface, this step will start as an out-of-band one, will collect the necessary information, then switch into being executed in-band by IPA. It will be equivalent to executing the existing `standby.prepare_image` command via the agent API, and will block until the image has been written. This allows us to remove this special case of command status polling. There will need to be a transition period to support old IPA ramdisks that do not support in-band deploy steps.

prepare_instance_boot

Priority: 60

This will be largely equivalent to the `prepare_instance_to_boot` method of the `AgentDeployMixin`.

tear_down_agent

Priority: 40

In this step, the node will be powered off and ramdisk boot environment will be removed.

switch_to_tenant_network

Priority: 30

In this step, the node will be switched from the provisioning network to tenant networks.

boot_instance

Priority: 20

In this step, the node will be powered on.

Agent heartbeat handler

The `heartbeat` method of the `HeartbeatMixin` currently provides an extension of the logic of the `deploy` step. This includes calling `continue_deploy` on the first heartbeat, and `reboot_to_instance` when the deployment is finished. This logic will be unnecessary with these methods as deploy steps, but will remain in place for a period for backwards compatibility. Drivers will advertise support for decomposed deploy steps by returning `True` from a method called `has_decomposed_deploy_steps`.

Proposed in-band deploy step support

In-band deploy steps will be handled in a similar way to in-band cleaning steps, with some differences:

- out-of-band deploy steps with a priority greater than 100 may be executed before the node has booted up
- the `deploy` interface may provide both in-band and out-of-band deploy steps
- there is no equivalent of manual cleaning
- IPA version mismatch will lead to termination of deployment

In-band deploy steps must have a priority between 41 and 99 to ensure they execute after `deploy` and before `tear_down_agent`.

In-band deploy steps will be driven through the agents heartbeat mechanism. The first heartbeat will query the in-band steps, combine them with out-of-band steps and store them in `driver_internal_info.deploy_steps`.

In-band deploy steps are advertised by the `deploy` interface, which will override the `get_deploy_steps` method to query the steps from IPA, and the `execute_deploy_step` method to execute them via the IPA API. This will be slightly different from clean steps, to support execution of out-of-band steps on the `deploy` interface. In-band deploy steps are always asynchronous, with polling triggered via the IPA heartbeat.

In-band deploy steps may request that the node is rebooted after completion of the step via a `reboot_requested` flag in their step definition. In the case where an IPA returns with a different version after reboot, deployment will be terminated.

Post-deploy step hooks will be supported via a `@post_deploy_step_hook` decorator, for example to set a nodes RAID configuration field.

The IPA ramdisk will be modified to add a new `deploy` extension. This will be very similar to the existing `clean` extension. Hardware managers should implement a `get_deploy_steps` method that should work in a similar way to the existing `get_clean_steps` method.

Alternatives

- Deny execution of out-of-band deploy steps before IPA has booted. See Observations for details.
- Allow in-band steps for boot from volume. These could be made available via an optional deploy step that boots up the node on the provisioning network.

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

RPC API impact

None

Driver API impact

There will be no changes to the driver API, but there will be changes to the `AgentDeployMixin` class used by all in-tree drivers and potentially out-of-tree drivers. These changes will be backwards compatible, with a transition period for out-of-tree drivers to switch to the new decomposed step model (advertised by returning `True` from `has_decomposed_deploy_steps`).

Nova driver impact

None

Ramdisk impact

Changes to the IPA ramdisk are discussed above. Backward compatibility will be provided by ignoring a missing `deploy` extension.

Security impact

In-band deploy steps will have additional access to the node by the nature of executing directly on it. These steps and the IPA ramdisk are under the control of the operator, who will need to take action to ensure that they do not introduce any security issues.

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

None

Developer impact

None

Implementation

Assignee(s)

Primary assignees:
Mark Goddard (mgoddard) Arne Wiebalck (arne_wiebalck)

Work Items

- Decompose core deploy step
- Advertise & execute in-band steps via IPA
- Collect & execute in-band steps from IPA
- Update documentation

Dependencies

- Deploy steps (implemented)
- Deploy templates (implemented)

Testing

Ideally tempest tests will be added to cover execution of in-band deploy steps. Software RAID configuration is a reasonable candidate for this as the resulting configuration could be verified.

Upgrades and Backwards Compatibility

This has been discussed elsewhere.

Documentation Impact

The deploy steps documentation will be updated, in particular covering the new flow and the required priorities of user steps.

References

None

5.3.2 15.2

Allow a ramdisk deploy user to specify their boot ISO

https://storyboard.openstack.org/#!/story/2007633

With support for virtual media, there are cases where an operator may wish to boot a machine with a specific virtual media image to facilitate the deployment of a machine or even just the completion of an action like firmware upgrades.

Providing an interface to signal boot to this iso, seems logical.

Problem description

A detailed description of the problem:

- The `ramdisk` deployment interface allows a user to define a kernel and ramdisk to be utilized to boot a ramdisk instance which after the initial deployment the instance is considered a deployed machine in active state.
- At present, because of the `ramdisk` deployment interface constraints, users are unable to specify ISOs for virtual media. They must supply a kernel/ramdisk and in the virtual media case it must be rebuilt.

Proposed change

- Allow a `instance_info/boot_iso` parameter to be leveraged to be the medium utilized for booting from an explicit ISO image which the conductor would support downloading, caching, and providing to the baremetal machine. This change would be focused on the virtual media boot interface code in relation to usage of Redfish.
- Teach the code related to the pass-through to provide the same basic capability to append parameters to the command line through decomposition of the ISO, appending to `grub2` and `isolinux` configurations with the supplied values, and repackaging of the ISO file for deployment.

Alternatives

- Use an external provisioning tool, and `adopt` the node into Ironic, if applicable.
- Pre-mastered machine specific configurations into ISO images which would ultimately result in pushing the preparation and execution workload to the API user.

Data model impact

None, this leverages the existing data model.

State Machine Impact

None

REST API impact

None, this change leverages existing JSON data fields with-in Ironics data model.

Client (CLI) impact

openstack baremetal CLI

None

openstacksdk

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

None

Security impact

None

Other end user impact

None

Scalability impact

The distinct possibility exists, if a user requests multiple concurrent deployments, that configuration injection could consume a large amount of disk space.

Also, we may wish to enable some sort of logic to prevent mass consumption of disk space as part of the conductor, for the reasons of cleanup, however the conductor has no real way to understand if this is a forever usage, or not. Ideally operator documentation would be updated to help scale planning for this condition. Alternatively we may wish to introduce a one-shot indicator flag so we dont attempt to persist ISOs after takeover on active machines.

Performance Impact

A large number of concurrent deployments may slow the conductor due to overall system performance constraints, depending on the exact options and settings leveraged.

Other deployer impact

None

Developer impact

None is anticipated, however we would likely focus on implementing this in the redfish virtual media code path, and should likely try to ensure that we do not make such changes redfish interface specific as other drivers are present in Ironic which support Virtual Media.

Implementation

Assignee(s)

Primary assignee:
Julia Kreger <juliaashleykreger@gmail.com>

Work Items

- Implement support to pass an explicit boot ISO into the ramdisk interface.
- Implement support to inject configuration into the boot ISO.
- Document this functionality for the ramdisk interface covering how to leverage this feature.

Dependencies

- None

Testing

Unit tests should be sufficient for ensuring this functionality is not broken.

A tempest test may also be viable, but we may wish to partner with the Metal3 community on integration testing, as ultimately this is essentially just an item of integration testing when virtual media AND ramdisk interfaces are leveraged.

Upgrades and Backwards Compatibility

N/A

Documentation Impact

We will want to update the documentation on the ramdisk deployment interface to detail this capability.

References

None

5.3.3 15.1

New Release Model for Ironic

This specification describes the new model of releasing deliverables under the ironic umbrella, including the Bare Metal service itself.

Problem description

Currently ironic follows the OpenStack release model *cycle-with-intermediary* that permits us to produce one or more releases per OpenStack cycle, with the last one in a cycle used as a final release as part of the OpenStack integrated release (called *named release* in this document, as it receives a code name). A stable branch is created from this final release for long-term bug fix support.

As the ironic community is exploring an increasing number of standalone applications, including ones outside of OpenStack (such as Metal3), one problem has become apparent: the stand-alone usage requires

not just frequent feature releases, but rather **supported** frequent feature releases. To be precise, we would like to produce supported releases every 1-3 months (2 months is what this spec proposes), where *supported* involves:

- Addressing at least critical issues with a point release. Currently we require consumers to switch to the next release, even if it is major.
- A supported upgrade path between releases. Currently only upgrades between OpenStack named releases are tested and supported, we need to support upgrades between intermediate releases as well.
- Documentation for each release. Currently only documentation for named releases (and master) is published.
- An obvious way to consume such releases. Currently deployment tools, including even Bifrost, are oriented on named releases or git master.

The proposed model aims to implement these requirements while keeping the existing commitments around the integrated release.

Proposed change

Terminology

The following concepts are used throughout this document:

named release
: is an integrated OpenStack release that receives a code name and a *named stable branch*.

intermediate release
: is a release of an ironic deliverable that happens as part of the master development (purposely excluding stable releases here) that does not correspond to a named release.

> **Note**
>
> This definition differs from the official OpenStack definition of an intermediary release. This is done on purpose to make the wording of this document clearer.

stable branch
: a branch created from a named release where bug fixes are applied and periodically released as *stable releases*.

stable release
: is a release created as part of stable support of a *named* release.

bugfix branch
: a branch created from an intermediate release that did NOT result in a stable branch.

Releasing

- Releases for all deliverables are evaluated on a loose bi-monthly basis, i.e. roughly every 2 months. If there are significant valuable changes, and a clear user interest in performing a release, we should do so. In most cases, this means a downstream distributor of Ironic is going to consume the release.

  This gives up to 6 releases a year, 3 per each OpenStack cycle.

- Two releases a year correspond to OpenStack named releases, the others happen between named releases. The former two happens always, the latter 4 can be skipped if there are no known consumers for that release or a deliverable does not see notable changes within 2 months.
- One week of soft feature freeze is observed before every release. *Soft* implies that feature can still merge if they are considered low-risk or critical for the scope of the upcoming release.

  This leaves merge windows of roughly 7-9 weeks for most features. Plans for them should be made during the feature freeze of the previous release.
- Ironic deliverables follow SemVer with one clarification: *patch* releases are always issued from a stable or bugfix branch (see *Stable branches and upgrades*). Releases from master always receive a minor or major version bump.

  Note

  This limitation is required to be able to find a suitable version for a branch release. E.g. imagine we cut 21.2.0 from master, then 21.2.1 also from master. If then we need to make a release from `stable/21.2` (the support branch for 21.2.0), there is no SemVer version to use (21.2.0.1 is still an option, of course, but may conflict with pbr).

- Intermediary (non-named) releases will target standalone users. OpenStack deployers will be recommended to use named releases.

Stable branches and upgrades

Service projects and bifrost

The following procedure will be applied to service projects (ironic and ironic-inspector), ironic-python-agent and bifrost:

- A stable branch is created from each release:
  - A traditional `stable/<code name>` branch for releases that coincide with named ones.
  - A `bugfix/<major.minor>` branch for other releases.

  The naming difference highlights the difference of intents:
  - *Stable* branches are designed for long-term consumption by downstreams (such as RDO) and for users to follow them.
  - *Bugfix* branches are a technical measure to allow patch releases after a certain release. Users and downstreams are not expected to follow them over a long period of time and are recommended to update to the next feature release as soon as it is out.
- Three upgrade paths are supported and tested in the CI for each commit:
  1. Between two subsequent named releases (e.g. *Train* to *Ussuri*).
  2. Between two subsequent intermediate releases.

     Note

     Its unlikely that well be able to use Grenade for that. Well probably use Bifrost instead.

3. From a named release to any intermediate release in the next release cycle.

   Note

   Supporting this path is technically required to implement CI for the other two paths).

Note

Operating CI on the non-named branches may require pinning devstack, tempest and ironic-tempest-plugin versions to avoid breakages. It will be determined on the case-by-case basis.

Other projects

Library projects (metalsmith, sushy, python-ironicclient and python-ironic-inspector-client) and networking plugins (networking-baremetal and networking-generic-switch) will be released and branched as before:

- Releases will be created on demand based on how many useful changes are available.
- Only named stable branches will be created, intermediate releases will not result in branching.

This procedure matches how libraries are usually released in the Python world.

The CI tools (virtualbmc and sushy-tools) and ironic-tempest-plugin will not be branched.

Support phases

- A named stable branch is supported according to the OpenStack policies, which is currently 1.5 years of full support followed by extended maintenance.
- Since this proposal significantly increases the number of branches in support, well tighten the rules around backports to named branches:
  - The first 12 months any bug fixes are acceptable. Low-risk features **may** be accepted if theyre believed to substantially improve the operator or user experience.
  - The last 6 months and during the extended maintenance phase only high and critical bug fixes are accepted.
  - If during the extended maintenance no changes merge to a branch within 6 months, this branch is considered abandoned and is closed for further backports.

    Note

    This also applies when the changes are proposed but cannot merge because of failing CI.

- Bugfix branches (for deliverables that have them) are supported for 6 months. Only high and critical bug fixes are accepted during the whole support time.

  Note

> It may mean that a stable branch created earlier will receive more fixes than a bugfix branch created later. This is a reflection of the fact that consumers are not expected to follow bugfix branches.

- As before, high and critical bug fixes **should** be backported to all supported branches once merged to master.

Dependencies

Dependencies handling for named releases and branches does not change. For example, we keep consuming upper-constraints of a corresponding branch.

For intermediate releases we will consume upper-constraints from a future named branch. E.g. for Victoria we would consume https://releases.openstack.org/constraints/upper/victoria.

The inter-service dependencies for both named and intermediate releases must be expressed separately, both via microversioning or via documentation. We already provide support for a broad set of versions of projects we can integrate with.

Deprecation policy

The deprecation policy remains intact: any deprecated functionality can only be removed after 6 months pass and a **named** release is done.

Alternatives

- Keep the current model, ask intermediate releases consumers to always upgrade to the latest one.

Data model impact

None

State Machine Impact

None

REST API impact

None

Microversioning is already used as a way to ensure cross-releases API compatibility.

Client (CLI) impact

None

openstack baremetal CLI

None

openstacksdk

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

We expect the Nova driver released as part of a certain OpenStack release series to be compatible *at least* with all Ironic releases from the same series and with the last release from the previous series.

Ramdisk impact

Under the proposed model, ironic, ironic-inspector and ironic-python-agent will get released at roughly the same time. The compatibility rules will be:

Each release of ironic/ironic-inspector is compatible with

- the release of ironic-python-agent that happens at the same time
- the last named release of ironic-python-agent

Note

Supporting releases between these two is very likely but is not officially guaranteed nor tested in the CI.

Each release of ironic-python-agent is compatible with

- the releases of ironic and ironic-inspector that happen at the same time
- the last named releases of ironic and ironic-inspector

Note

The first 3 rules are already enforced in the CI, the last will require a new job on ironic-python-agent, supposedly based on Bifrost.

The compatibility matrix will be provided through the documentation as part of the pre-release documentation update and via the future web site.

We will publish ironic-python-agent images corresponding to all stable branches, named and intermediate (currently images are only published for named branches) and provide instructions on how to build customized images based on a certain branch or release.

Security impact

Supported intermediate releases will also receive security bug fixes.

Other end user impact

See *Other deployer impact*.

Scalability impact

None

Performance Impact

None

Other deployer impact

Deployers will have faster access to new features if they opt for using intermediate releases.

Developer impact

No direct impact. The *Deprecation policy* is not changed.

Implementation

Assignee(s)

The whole team is expected to be responsible for executing this plan, the primary assignee(s) will coordinate it.

Primary assignee:
Dmitry Tantsur (@dtantsur, dtantsur@protonmail.com)

Work Items

- Discuss this document with the release team and the TC. Make necessary adjustments to our deliverables in the release repository.
- Update the releasing documentation and publish our release schedule.
- Create new CI jobs as described in *Testing*.
- Start publishing ironic-python-agent images from non-named stable branches (may work out-of-box).
- Update Bifrost to support installing components from latest published releases.

Dependencies

None

Testing

Two new family of the CI jobs will be introduced:

- Intermediary upgrade jobs on ironic and ironic-inspector, testing upgrade from the last intermediate release branch.
- Backwards compatibility job on ironic-python-agent to test every commit against the previous named releases of ironic and ironic-inspector (e.g. during the Victoria cycle ironic-python-agent is tested against stable/ussuri of ironic and ironic-inspector).

Third party CI jobs are expected to run on the intermediate branches the same way as they would on master. As soon as support for a specific branch is over, the 3rd party CI jobs may be turned off for it. Since we are only going to accept high and critical bug fixes to new branches, only minor load increase is expected on 3rd party CI systems.

Upgrades and Backwards Compatibility

See *Stable branches and upgrades* and *Testing*.

Documentation Impact

To make intermediate releases obviously consumable, we will need a new web site focused around standalone ironic. It will display the latest versions of the components and ironic-python-agent images, point at the way to consume them and provide documentation for each minor or major release.

The releasing documentation will be updated to follow this model.

References

5.4 Ussuri

5.4.1 15.0

Indicator management

https://storyboard.openstack.org/#!/story/2005342

This spec proposed creating of API for hardware indicators management.

Problem description

Depending on the hardware vendor, bare metal machines might carry on-board and expose through the baseboard management controller (BMC) various indicators, most commonly LEDs.

For example, a blade system might have LEDs on the chassis, on the blades (compute cards), on the drives, on the PSUs, on the NICs.

The use-cases when ironic-manageable LEDs might make sense include:

- The DC staff member want to identify hardware unit in the rack by lighting up its LED in ironic
- The Deployer wants to identify ironic node by pressing a button on the physical unit to light up the LED
- The remote Admin user wants to be aware of the failure LEDs lighting on the bare metal nodes possibly indicating a failure

Proposed change

Overview

This RFE proposes an extension of the node ReST API endpoint that will allow reading and toggling the indicators (e.g. LEDs) on the hardware units.

Indicator management workflow

1. An API client can choose to discover available node indicators by sending `GET /v1/nodes/<node_ident>/management/indicators` request. This step is optional.
2. An API client reads the indicator on the chosen component of the bare metal node by sending `GET /v1/nodes/<node_ident>/management/indicators/<component>/<ind_ident>` request and presenting current indicator state to the user.
3. The API client can change the state of the indicator(s) on the given component by sending `PUT /v1/nodes/<node_ident>/management/indicators/<component>` request.

Alternatives

The user can communicate with BMC independently from ironic for the same purpose. Though ironic node correlation with physical node may be challenging.

Data model impact

None.

Due to the interactive nature of indicator information, it seems that user-indicator link should be as immediate as possible, having it cached by the database can make indicators confusing.

State Machine Impact

None.

Indicators should always work the same regardless of the machine state.

REST API impact

- `GET /v1/nodes/<node_ident>/management/indicators`

  Retrieves bare metal node components. Returns a JSON object listing all available node components.

  Currently known components are: `system`, `chassis` and `drive`. Indicator names are free-form, but hopefully descriptive.

  Error codes:

  - 404 Not Found if node is not found.

  Example response object:

```
{
    "components": [
        {
```

(continues on next page)

(continued from previous page)


```
            "name": "system",
            "links": [
                {
                    "href": "http://127.0.0.1:6385/v1/nodes/Compute0/
                    management/indicators/system",
                    "rel": "self"
                },
                {
                    "href": "http://127.0.0.1:6385/nodes/Compute0/
                    management/indicators/system",
                    "rel": "bookmark"
                }
            ]
        },
        {
            "name": "chassis",
            "links": [
                {
                    "href": "http://127.0.0.1:6385/v1/nodes/Compute0/
                    management/indicators/chassis",
                    "rel": "self"
                },
                {
                    "href": "http://127.0.0.1:6385/nodes/Compute0/
                    management/indicators/chassis",
                    "rel": "bookmark"
                }
            ]
        }
    ]
}
```


- `GET /v1/nodes/<node_ident>/management/indicators/<component>`

  Retrieves indicators for a component. Returns a JSON object listing all available indicators for given hardware component along with their attributes.

  Currently known components are: `system`, `chassis` and `drive`. Indicator names are free-form, but hopefully descriptive.

  Error codes:

  - 404 Not Found if node or component is not found.

  Example response object:


```
{
    "indicators": [
        {
            "name": "power",
            "readonly": true,
            "states": [
```


(continues on next page)

(continued from previous page)


```
                "OFF",
                "ON"
            ],
            "links": [
                {
                    "href": "http://127.0.0.1:6385/v1/nodes/Compute0/
                    management/indicators/system/power",
                    "rel": "self"
                },
                {
                    "href": "http://127.0.0.1:6385/nodes/Compute0/
                    management/indicators/system/power",
                    "rel": "bookmark"
                }
            ]
        },
        {
            "name": "alert",
            "readonly": false,
            "states": [
                "OFF",
                "BLINKING",
                "UNKNOWN"
            ],
            "links": [
                {
                    "href": "http://127.0.0.1:6385/v1/nodes/Compute0/
                    management/indicators/system/alert",
                    "rel": "self"
                },
                {
                    "href": "http://127.0.0.1:6385/nodes/Compute0/
                    management/indicators/system/alert",
                    "rel": "bookmark"
                }
            ]
        },
    ]
}
```


- `GET /v1/nodes/<node_ident>/management/indicators/<component>/<ind_ident>`

  Retrieves indicator state for the component. Returns a JSON object representing current state of the chosen indicator (`ind_ident`) sitting on the `component`.

  The field of the response object is `state`, the value is one of: `OFF`, `ON`, `BLINKING` or `UNKNOWN`.

  Error codes:

  - 404 Not Found if node, component or indicator is not found.

  Example response object:

```
{
  "state": "ON"
}
```

- `PUT /v1/nodes/<node_ident>/management/indicators/<component>/<ind_ident>`

  Set the state of the desired indicators of the component. The endpoint accepts a JSON object. The following field is mandatory:

  - `state` requested indicator state
  - 400 Bad Request if `state` is not an accepted value
  - 404 Not Found if node, component or indicator is not found.

  Example request object:

```
{
  "state": "ON"
}
```

Client (CLI) impact

ironic CLI

None.

openstack baremetal CLI

The following commands will be created:

```
openstack baremetal node indicator list <node> [component]
openstack baremetal node indicator show <node> <component> indicator
openstack baremetal node indicator set <node> <component> <indicator> --state
↪{ON,OFF,BLINKING}
```

The first command lists all indicators for the specified component, or for all components if specific component is not given.

RPC API impact

The new RPC calls are introduced:

- Listing the indicators

```
def get_supported_indicators(self, context, node_id, component=None):
    """Get node hardware components and their indicators.

    :param context: request context.
    :param node_id: node id or uuid.
    :param component: The hardware component, one of
        :mod:`ironic.common.components` or `None` to return all
        available components.
    :returns: a `dict` holding indicator IDs as keys, indicator properties
```

(continues on next page)

(continued from previous page)

```
        as values. Indicator properties is a `dict` that includes:
        `readonly` bool, `states` list containing zero or more values from
        mod:`ironic.common.indicator_states`.
    """
```

- Reading the indicator

```
def get_indicator_state(self, context, node_id, component, indicator):
    """Get node hardware component indicator state.

    :param context: request context.
    :param node_id: node id or uuid.
    :param component: The hardware component, one of
        :mod:`ironic.common.components`.
    :param indicator: Indicator IDs, as
        reported by `get_supported_indicators`
    :returns: current indicator state. One of the values from
        mod:`ironic.common.indicator_states`.
    """
```

- Setting the indicator

```
def set_indicator_state(self, context, node_id, component,
                        indicator, state):
    """Set node hardware components indicator to the desired state.

    :param context: request context.
    :param node_id: node id or uuid.
    :param component: The hardware component, one of
        :mod:`ironic.common.components`.
    :param indicator: Indicator IDs, as
        reported by `get_supported_indicators`)
    :param state: Indicator state, one of
        mod:`ironic.common.indicator_states`.
    """
```

Driver API impact

Optional indicator API methods is added to *ManagementInterface*:

- Listing the indicators

```
def get_supported_indicators(self, task, component=None):
    """Get a map of the supported indicators (e.g. LEDs).

    :param task: A task from TaskManager.
    :returns: A dictionary of hardware components
        (:mod:`ironic.common.components`) as keys with indicator
        properties as values. Indicator properties is a `dict`
        that includes: `readonly` bool, `states` list containing
```

(continues on next page)

(continued from previous page)

```
        zero or more values from mod:`ironic.common.indicator_states`.
    """
```

- Reading the indicator

```
def get_indicator_state(self, task, component, indicator):
    """Get current state of the indicator of the hardware component.

    :param task: A task from TaskManager.
    :param component: The hardware component, one of
        :mod:`ironic.common.components`.
    :param indicator: Indicator ID (as reported by
        `get_supported_indicators`).
    :returns: current indicator state. One of the values from
        mod:`ironic.common.indicator_states`.
    """
```

- Setting the indicator

```
def set_indicator_state(self, task, component, indicator, state):
    """Set indicator on the hardware component to the desired state.

    :param task: A task from TaskManager.
    :param component: The hardware component, one of
        :mod:`ironic.common.components`.
    :param indicator: Indicator ID (as reported by
        `get_supported_indicators`).
    :state: Desired state of the indicator, one of
        :mod:`ironic.common.indicator_states`.
    """
```

The above methods are implemented for Redfish and IPMI hardware types.

Nova driver impact

None.

Ramdisk impact

None.

Security impact

None.

Other end user impact

The indicators can be made accessible through Horizon or other UI tools.

Scalability impact

None.

Performance Impact

None.

Other deployer impact

None.

Developer impact

None.

Implementation

Assignee(s)

Primary assignee:
<etingof>

Work Items

- Add indicator management methods to ironic management interface
- Add indicator management to ironic ipmi and redfish hardware types
- Add RPC for indicator management
- Add REST API endpoint for indicator management

Dependencies

None.

Testing

- Unit tests and Tempest API will be provided

Upgrades and Backwards Compatibility

This change is fully backward compatible.

Documentation Impact

API reference will be provided.

References

Allow Leasable Nodes

https://storyboard.openstack.org/#!/story/2006506

A large bare metal deployment may consist of hardware owned by multiple owners who lease their nodes to users - lessees - who gain temporary and limited access to specific nodes. Ironic understands the concept of a hardware owner: a node can set its `owner` field to a project, and that project can gain API access to that node through the use of an updated policy file. However, Ironic does not understand the concept of a node lessee.

This spec describes a solution that accommodates the notion of a node lessee.

Problem description

Ironic currently supports two classes of users: administrators, who control the entire inventory of hardware; and owners, who have policy-limited API access to the nodes that they own. However, Ironic does not support non-administrative users - users who can deploy upon a node, and perhaps have access to an extremely limited subset of the API (such as node power functions).

Proposed change

Node lessees can be supported with the following changes:

- Add a new `lessee` field to the node object. This field must either be empty or set to a Keystone project id.
- Update the node controller so that policy checks pass in a nodes `lessee` information. Note that Ironic already does that with node owners[0].
- Update Ironics default generated policy file to include an `is_node_lessee` rule:
  - is_node_lessee: project_id:%(node.lessee)s

  The remainder of the policy file will stay the same, so that there is no change to default API access.
- Update the node list function so that projects with access to `baremetal:node:list` are returned nodes with matching `owner` or `lessee` fields.
- Update Ironic allocations so that allocations with owners can match nodes by a nodes `owner` or `lessee`.

Note that this work does not add any new scheduling responsibilities in Ironic. A new Nova filter, such as an updated version of the proposed NodeOwnerFilter[1], would be desirable; and Blazar could integrate with the `lessee` field as they see fit. However, the proposed work does integrate well with the existing ability to create a restricted allocation.

Further down the line when Ironic creates a Deployment API, we can have the new Deployment API actions default to being accessible to node lessees.

Alternatives

Lessee information could be stored in a dictionary field such as `properties` or `extras`. However this makes updating database queries far more difficult, and the non-administrative user concept feels distinct enough to warrant a new field.

[0] https://opendev.org/openstack/ironic/src/branch/master/ironic/api/controllers/v1/utils.py#L1178
[1] https://review.opendev.org/#/c/697331/

Data model impact

A `lessee` field will be added to the nodes table as a `VARCHAR(255)` with a default value of `null`.

State Machine Impact

None

REST API impact

- A `lessee` field will be returned with the node object.
- The REST API will pass in the `lessee` for node policy checks.
- The API will be updated to allow a user to set/unset the value through the API.
- The node list API will be updated to allow filtering by `lessee`.
- The limited `baremetal:node:list` action will be updated to match nodes by both lessee and owner.
- A new API microversion will be introduced for the new node `lessee` field.

Client (CLI) impact

None

ironic CLI

None

openstack baremetal CLI

An update will be needed to enable a user to set/unset `lessee` from the command line.

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

None

Security impact

This change allows functionality to be exposed to additional users. However this access is blocked off by default; it requires an update to the Oslo policy file, and can be adjusted as an administrator desires.

Other end user impact

None

Scalability impact

None

Performance Impact

Some functionality that previously matched nodes by `owner` will now have to match both `owner` and `lessee`. This should be doable at the database query level.

Other deployer impact

None

Developer impact

None

Implementation

Assignee(s)

Primary assignees: * tzumainn - tzumainn@redhat.com

Work Items

- Add database field.
- Add object field.
- Add REST API functionality and microversion.
- Update REST API documentation.
- Update python-ironicclient.
- Update node controller.
- Update allocations conductor.
- Write tests.
- Write documentation detailing usage.

Dependencies

None

Testing

We will add unit tests and Tempest tests.

Upgrades and Backwards Compatibility

The `lessee` node field will be created as part of the upgrade process with a default value in the database schema. This change has no end-user impact if the policy file is not updated.

Documentation Impact

REST API documentation will be updated.

References

Support for node retirement

https://storyboard.openstack.org/#!/story/2005425

This spec proposes to add initial support for retiring nodes from ironic.

Retiring nodes is a natural part of a servers life cycle, for instance when the end of the warranty is reached and the physical space is needed for new deliveries to install replacement capacity.

However, depending on the type of the deployment, removing nodes from service can be a full workflow by itself as it may include steps like moving applications to other hosts, cleaning sensitive data from disks or the BMC, or tracking the dismantling of servers from their racks.

In order to help Deployers with these tasks, ironic should provide basic means to mark nodes as being retired, i.e. mark them as not eligible for scheduling any longer (while still allowing other operations such as cleaning), or support a convenient search for and list such nodes.

Extensions of this initial support, such as extending the state machine by an explicit retired state (which could be used as a starting point for a dedicated retirement workflow with end-of-life cleaning or other more elaborate actions, such as prepare for donation) are beyond the scope of this spec.

Problem description

There is currently no explicit support in ironic for node retirement: when a node needs to be taken out of service, the instance is deleted (which triggers cleaning) and the node is deleted, either directly from state available or after being moved to manageable (and potential additional cleaning).

There are at least two issues that this spec tries to fix:

- between the deletion of the instance and the deletion of the node, there is a time window in which a new instance could be scheduled to a node on its way into retirement (setting maintenance to True does only partially help with this scheduling race as apart from not being meant to be used in this situation this will prevent cleaning);
- there is no easy way to mark nodes as retired; this makes identifying and listing such nodes (as required by third party tools or the remaining retirement workflow) cumbersome.

The retirement use case is different from other use cases, like a non-functional node which needs to be taken out of service for some time, in that nodes may be marked for retirement a long time before anything actually happens on the node. Also, retired nodes are not supposed to enter service again.

Proposed change

The proposal is to extend the list of node properties by two new fields:

- retired (True/False)
- retired_reason (text)

in analogy to the existing protected and protected_reason.

The retired field shall signal that a node is meant to be retired and that the node should not be considered for scheduling any longer. The retired field can be set irrespective of the nodes state, in particular when the node is active.

Active nodes which are cleaned while retired is True, e.g. upon instance deletion, go to manageable (rather than available). This leaves no window where a retired node would receive another instance. Otherwise, retired set to True shall not interfere with cleaning or rebuilding.

Nodes with retired set to True cannot move from manageable to available (to prevent accidental reuse): the provide verb is blocked. In order to move these nodes to available, the retired field needs to be set to False first.

The new field shall also be used to get these nodes quickly via a list command, e.g. by an additional flag retired (in analogy to maintenance).

Alternatives

An alternative to address the lack of support for node retirement would be to introduce a new state retired in the ironic state diagram. While this would require additional efforts to implement, there are no obvious benefits compared to the proposal of this specification.

Using the currently available means in ironic, such as the maintenance state and maintenance_reason, is certainly possible but will cause inconveniences during node cleaning and when trying to extract the list of nodes to be removed.

Data model impact

The nodes table will get two additional fields:

- retired (tinyint)
- retired_reason (text)

For existing instances, i.e. for data migration, retired will be set to False and retired_reason will be left empty.

State Machine Impact

The retired field can be set on nodes in any state except available. Nodes in available should be moved to manageable first to ensure backwards compatibility with tools like metalsmith. This also eliminates the need for changes in the nova driver.

The retired field can also be set for nodes in transient states, as long as the node is not locked.

When retired is set to True, a node will not move to available after cleaning, but to manageable.

When retired is set to True, a node cannot move from manageable to available, the corresponding provide verb will return an HTTP 409.

REST API impact

The REST API will need to be extended with a new version to include the new retired field, in a similar way as the protected field was introduced [0][1].

In addition, there should be a filter /v1/nodes?retired=True/False to easily identify retired nodes.

Client (CLI) impact

ironic CLI

None.

openstack baremetal CLI

The retired and retired-reason options will be added to the set and unset subcommands:

- openstack baremetal node set retired retired-reason <reason> <node>
- openstack baremetal node unset retired <node>

shall be added to set and unset a nodes retired field (and provide a reason in the case of set).

An additional flag retired shall be added to the openstack baremetal node list command to restrict the returned result to the nodes which have the retired flag set.

RPC API impact

Driver API impact

None.

Nova driver impact

None.

Ramdisk impact

None.

Security impact

None.

Other end user impact

None.

Scalability impact

None.

Performance Impact

None.

Other deployer impact

None.

Developer impact

None.

Implementation

Assignee(s)

Primary assignees:
Arne Wiebalck (arne_wiebalck) Riccardo Pittau (rpittau)

Other contributors: None

Work Items

- Add new database fields
- **Adapt state machine if retired is set to True**
  - change cleaning > manageable
  - move available > manageable
  - block manageable > available
- Exclude retired nodes from periodic tasks
- Extend API
- Extend cli
- Add new API to openstacksdk
- Add documentation

Dependencies

None.

Testing

Needs testing similar to maintenance.

Upgrades and Backwards Compatibility

Upon upgrades the new fields need to be set as specified in the Data model impact section.

Documentation Impact

The additional retired field and its intended use need to be documented.

References

[0] https://storyboard.openstack.org/#!/story/2003869 [1] https://review.opendev.org/#/c/611662

5.4.2 14.0

Boot and network management for in-band inspection

https://storyboard.openstack.org/#!/story/1584830 https://storyboard.openstack.org/#!/story/1528920

This spec suggests making Ironic Inspector play well with the tenant network separation and non-PXE boot interfaces.

Problem description

With the `neutron` network interface nodes are no longer constantly connected to the provisioning network. We need to connect them manually before in-band inspection, which is inconvenient and error-prone.

This change covers integration with both boot and network interfaces.

Proposed change

The proposed flow will work as follows:

1. Inspection with the `inspector` inspect interface is started via the API.
2. The `inspector` inspect interface:
    1. Calls `task.driver.network.validate_inspection`.

       If it raises `UnsupportedDriverExtension`, fall back to the code path.
    2. Calls `task.driver.boot.validate_inspection`.

       If it raises `UnsupportedDriverExtension`, fall back to the code path.
    3. Calls `task.driver.network.add_inspection_network`. It creates a port on the `inspection_network`.
    4. Calls `task.driver.boot.prepare_ramdisk` providing kernel parameters from the option `[inspector]extra_kernel_params`.
    5. Calls the ironic-inspector introspection API with `manage_boot=False`.
    6. Powers on the machine via `task.driver.power`.
3. Now inspection proceeds as previously.

Boot and network interfaces

- Add a new call `validate_inspection`. It will be implemented the same way as `validate_rescue`, but instead of raising `MissingParameterValue` on absent parameters it will raise `UnsupportedDriverExtension` to indicate fall back to the old approach.
  - Implement `validate_inspection` for the PXE and iPXE boot interfaces.
- Add a new `driver_info` parameter `driver_info[inspection_network]` and a new configuration option `[neutron]inspection_network`.
- Extend the `NetworkInterface` to provide `add_inspection_network`, `remove_inspection_network` and `validate_inspection` similarly to rescue networks. However, `validate_inspection` will raise `UnsupportedDriverExtension` if the inspection network is not specified.

Inspector inspect interface

Modify the `Inspector` inspect interface to follow the flow outlined above.

- Call `boot.validate_inspection` and `network.validate_inspection` in the beginning of the introspection process. If either raises `UnsupportedDriverExtension`, follow the same procedure as previously.
- Call `network.add_inspection_network` before and `network.remove_inspection_network` after inspection.
- Add a new `driver_info` parameter `driver_info[inspector_extra_kernel_params]` and a new configuration option `[inspector]extra_kernel_params`.
- Call `boot.prepare_ramdisk` before introspection, providing the ironic-inspector URL (fetched from the service catalog) and `extra_kernel_params` to the `ramdisk_params` argument. Call `boot.cleanup_ramdisk` afterwards.
- Call ironic-inspector passing `manage_boot=False`.

Inspecting ports

Currently Ironic Inspector does not require ports, port groups or local link information to be present to conduct inspection. However, to use network flipping we will need this information, which can be:

- entered manually by an operator (using out-of-band inspection if possible) OR
- inspected initially with a node manually put on the right network.

Alternatives

- Do not support network separation.
- Expose the network and boot interfaces in Ironic API and make Inspector use it.

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

ironic CLI

None

openstack baremetal CLI

None

RPC API impact

None

Driver API impact

Extend the `NetworkInterface` with:

```
def validate_inspection(self, task):
    """Validates the network interface for inspection operation.

    :param task: A TaskManager instance.
    :raises: InvalidParameterValue, if the network interface configuration
        is invalid.
    :raises: MissingParameterValue, if some parameters are missing.
    """
    raise exception.UnsupportedDriverExtension(
        driver=task.node.driver, extension='validate_inspection')

def add_inspection_network(self, task):
    """Add the inspection network to a node.

    :param task: A TaskManager instance.
    :raises: NetworkError
    """
    pass

def remove_inspection_network(self, task):
    """Remove the inspection network from a node.

    :param task: A TaskManager instance.
    """
    pass
```

Extend the `BootInterface` with:

```
def validate_inspection(self, task):
    """Validate that the node has required properties for inspection.

    :param task: A TaskManager instance with the node being checked
    :raises: MissingParameterValue if node is missing one or more required
        parameters
    :raises: UnsupportedDriverExtension
    """
    raise exception.UnsupportedDriverExtension(
        driver=task.node.driver, extension='validate_inspection')
```

Nova driver impact

None

Ramdisk impact

None

Security impact

This change will also allow using in-band inspection with tenant network separation increasing security.

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

New configuration options:

- `[neutron]inspection_network` the default inspection network (no default).
- `[inspector]extra_kernel_params` the default kernel parameters to pass to introspection (empty by default).

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
Dmitry Tantsur (lp: divius, irc: dtantsur)

Work Items

1. Add new methods to the network and boot interfaces.
2. Update the `inspector` inspect interface to use them.

Dependencies

None

Testing

Coverage by unit tests.

Upgrades and Backwards Compatibility

The default behavior will not change because the `inspection_network` will be left unpopulated initially. After it gets populated, nodes with ports will follow the new flow for introspection. This feature can be enabled per node by setting `inspection_network` on nodes, not globally.

This work does not anyhow affect introspection that is started using the ironic-inspectors own CLI or API.

Documentation Impact

The Ironic documentation should be updated to explain using network separation with in-band inspection.

References

Allow Node Owners to Administer Nodes

https://storyboard.openstack.org/#!/story/2006506

This spec describes an update that allows a node owner to administer their node through the Ironic API without being able to administer the entire node cluster. This is accomplished by exposing owner node data in the REST API for policy checks.

Problem description

Ironic is not multi-tenant; anyone with API access to one node has access to all nodes. While nodes have an `owner` field, it is purely informational and not tied into any form of access control.

Bringing full multi-tenant support to Ironic would allow us to address the following user stories:

Standalone Ironic

As users of a shared data center, we would like to use Ironic to manage our hardware resources. Each work group should be able to control their own resources without having access to hardware resources owned by other groups.

Data center operator

As the operator of shared datacenter, I would like to delegate power control of baremetal hardware to the hardware owners using the Ironic API. By using the Ironic API instead of existing embedded management protocols (such as IPMI) I can maintain the simplicity of a shared management network while having granular control over who can access what hardware.

Proposed change

The Ironic API already has a way of controlling access to the REST API: a policy engine[0] that is used throughout the REST API. However, when the node controller uses the policy engine[1], it does so without passing in any information about the node being checked. Other services such as Nova[2] and Barbican[3] pass in additional information, and we propose doing the same. In summary, we would like to:

- Assume that a nodes `owner` field is set to the id of a Keystone project. We refer to this project as the node owner.
- Update the node controller so that policy checks also pass in information about the node, including the `owner` field.
- Update Ironics default generated policy file to include an `is_node_owner` rule:
  - is_node_owner: project_id:%(node.owner)s

  The remainder of the policy file would stay the same, meaning that there is no change to default API access.
- Create documentation explaining how to update an Ironic policy file to give API access to owners. For example:
  - baremetal:node:set_power_state: rule:is_admin or rule:is_node_owner

  Note that Nova grants API access in a similar manner[4].

This update is enough to control access for most API functions - except for list functions. For those, we propose the following:

- Right now, the policy rule baremetal:node:get is used for both `get_all` and `get_one`. We can add two new policy rules: `baremetal:node:list_all` for retrieving all nodes, and `baremetal:node:list` for retrieving nodes whose `owner` field matches the querying project.
- Updating the REST API node list functions to first perform a policy check against `baremetal:node:list_all`. If it passes, then we return all nodes; otherwise we perform a pol-

[0] https://github.com/openstack/ironic/blob/master/ironic/common/policy.py

[1] https://github.com/openstack/ironic/blob/master/ironic/api/controllers/v1/node.py#L225 Example of a current policy check. Note the use of `cdict`; it is being passed in as both the `target` and the `creds`.

[2] https://github.com/openstack/nova/blob/master/nova/api/openstack/compute/servers.py#L648-L652 Example of Nova creating a `target` dictionary.

[3] https://github.com/openstack/barbican/blob/stable/rocky/barbican/api/controllers/__init__.py#L59-L72 Example of Barbican creating a `target` dictionary.

[4] https://github.com/openstack/nova/blob/master/nova/policies/base.py#L27-L30 Example of Nova defaulting a rule that uses information from a `target` dictionary.

icy check against `baremetal:node:list`. If that passes, then we return nodes filtered by the `owner` field against the project specified in the request context. These list functions already have the option of filtering by `owner`[5].

- Updating the default generated policy file to set `baremetal:node:list_all` and `baremetal:node:list` to `rule:baremetal:node:get` to ensure backwards compatibility.

Allocation API changes

Allocation objects represent a request to find and reserve a suitable node, and as such should be available for non-administrator users. For this purpose we will introduce an `owner` field for allocations and start distinguishing two types of allocations:

unrestricted allocations
: work the same as before. They can be created only by administrative users and can have any `owner` set. Creating them is governed by the existing policy `baremetal:allocation:create`.

restricted allocations
: are created by non-administrative users and have `owner` matching the project ID of a creating user. Creating them will be governed by a new policy `baremetal:allocation:create_restricted`.

The two policies will be exclusive and work as follows:

1. First, `baremetal:allocation:create` is checked. If it passes, then any value of `owner` is accepted. If no `owner` is provided, the field is left as `None`.
2. If the first check does not pass, `baremetal:allocation:create_restricted` is checked. The `owner` field must match the project ID of the user or be empty (in which case it is set to the project ID of the user). HTTP Forbidden is returned if:
    1. The project ID of the user cannot be determined (i.e. restricted allocations do not work in the standalone mode).
    2. The requested non-empty owner does not match the project ID.
3. Otherwise, creating the allocation fails.

Finally, when processing an allocation with non-empty `owner`, only nodes with a matching `owner` are considered.

Alternatives

One alternative is to perform these checks at the database API level. However that requires a new code pattern to be used on a large number of individual functions.

Data model impact

None

[5] https://github.com/openstack/ironic/blob/master/ironic/api/controllers/v1/node.py#L1872

State Machine Impact

None

REST API impact

See details in Proposed change above.

Client (CLI) impact

None

ironic CLI

None

openstack baremetal CLI

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

None

Security impact

This change allows functionality to be exposed to additional users. However this access is blocked off by default; it requires an update to the Oslo policy file, and can be adjusted as an administrator desires.

Other end user impact

None

Scalability impact

None

Performance Impact

None: although node data needs to be retrieved in order to pass that information into a policy check, the controller functions already fetch that information.[6]

Other deployer impact

None

Developer impact

None

Implementation

Assignee(s)

Primary assignees: * tzumainn - tzumainn@redhat.com * larsks - lars@redhat.com

Other contributors: * dtantsur - dtantsur@redhat.com

Work Items

- Update node controller.
- Add documentation.
- Write tests.

Dependencies

None

Testing

We will add unit tests and Tempest tests.

Upgrades and Backwards Compatibility

Existing Ironic installations that use the `owner` field for something other than a project ID will be minimally affected for two reasons:

- If the `owner` field does not match a project ID (or is None), the proposed update to the policy file will not give any non-admin access to the Ironic API.
- This change has no end-user impact if the policy file is not updated. An existing install can simply choose not to update their policy file.

Documentation Impact

We will include additional documentation describing the possible applications of using the `node_owner` policy roles.

[6] https://github.com/openstack/ironic/blob/master/ironic/api/controllers/v1/node.py#L227

References

5.5 Train

5.5.1 13.0

Define idrac Hardware Type Support of Redfish Interfaces

https://storyboard.openstack.org/#!/story/2004592

Operators need the ability to configure Ironic to use Redfish to manage Dell EMC bare metal servers, and be assured it:

- offers a means of incrementally increasing the use of Redfish as Ironics support, the standard, and Dell EMC service implementations evolve,
- delivers management protocol choice among those supported by Dell EMC Intelligent Platform Management Interface (IPMI), Redfish, and Web Services Management (WS-Man),
- provides all the `idrac` hardware type functionality they have relied on,
- works and will continue to,
- is supported by the vendor and community, and
- can offer Dell EMC value added.

This specification suggests the `idrac` hardware type provides that.

Problem description

Use cases

Expanding on what the introductory paragraph describes above, several use cases can be envisioned. While this specification enables them, it may turn out only some will find practical use among operators. That could be driven by many factors, including an existing versus greenfield deployment, operator comfort level with Redfish versus WS-Man, maturity of the protocol implementations, availability of needed functionality across Redfish and WS-Man, requirement for vendor value added, operational plans and schedules, among others.

Here they are. Note that except for the first two, they can be used in combination to configure a single Dell EMC bare metal server.

- An Admin User has a Dell EMC bare metal server and uses only WS-Man to manage it.
- An Admin User has a Dell EMC bare metal server and uses only Redfish to manage it.
- An Admin User has a Dell EMC bare metal server and uses both Redfish and WS-Man to manage it. When both offer the needed functionality, either is used.
- An Admin User has a Dell EMC bare metal server and uses both Redfish and WS-Man to manage it. Ironics ability to manage the server is maximized.
- An Admin User has a Dell EMC bare metal server and uses both Redfish and WS-Man to manage it. Vendor value added, which is available from only one or both, is used.

Proposed change

Background

The `idrac` hardware type is the Ironic driver intended for use with Dell EMC bare metal servers equipped with an integrated Dell Remote Access Controller (iDRAC) baseboard management controller (BMC). To date, all the out-of-band (OOB) management protocol-dependent interface implementations `idrac` has supported use the WS-Man protocol to interact with the iDRAC. Those implement the `inspect`, `management`, `power`, `raid`, and `vendor` hardware interfaces. Like the hardware type, they are named `idrac`. They rely on python-dracclients WS-Man client.

Operators also have the option to use the generic, vendor-independent `redfish` hardware type with Dell EMC bare metal servers that have an iDRAC that supports the Redfish protocol. `redfish`s supported OOB protocol- dependent interface implementations use the Redfish protocol to interact with the BMC. Those implement the `bios`, `inspect`, `management`, and `power` hardware interfaces. Again, like the hardware type, they are named `redfish`. They rely on sushys Redfish client. Importantly, while some of those work with the iDRAC, including `management` and `power`, not all of them do.

The `redfish` hardware type enables managing servers compliant with the Redfish protocol. However, it is relatively new, and the protocol standard has been evolving, along with its implementations by hardware vendors such as Dell EMC. As is common among standards, there is a difference between compliance and interoperability. For example, the Redfish `bios` interface implementation has not worked with the iDRAC because of client and server protocol implementation incompatibility.

While there is much functional overlap between the interface implementations supported by the `idrac` and `redfish` hardware types, it is not complete. Only `idrac` supports a `raid` interface implementation and only `redfish` supports `bios`. Also, the optional hardware interface functionality available in the `idrac` and `redfish` interface implementations can differ. For example, while the `redfish` implementation of the `management` hardware interface first introduced optional boot mode functionality, `idrac` does not offer that, yet. Therefore, those two hardware types are not perfect substitutes for one another.

Dell EMC wants to be able to offer its customers vendor value added as supported by the Redfish standard, like it has done through WS-Man. That benefits operators by making available features and functionality that has not yet been standardized. Dell EMC can be more responsive to its customers needs and differentiate itself in the market.

Goal

With this specification, we are going to achieve the goal of promoting and accelerating the adoption of Redfish by operators with Dell EMC bare metal servers.

Non-goals

The following is considered outside the scope of this specification:

- Support a node configuration with a mix of Redfish and WS-Man `management` and `power` interface implementations. The legacy `idrac` implementations of the `management` and `power` hardware interfaces interact to set the boot device. It is not clear there is a compelling need to accommodate that in a mixed Redfish and WS-Man configuration.
- The following TripleO command can be used to register and configure nodes for their deployment with Ironic:

```
openstack overcloud node import instackenv.json
```

See the TripleO register nodes documentation. It sets properties in a nodes `driver_info` field which are required by its driver. Presently, when the nodes driver is `idrac`, those are the properties `drac_address`, `drac_username`, and `drac_password` needed by the WS-Man interface implementations `idrac` has supported. See the iDRAC driver documentation.

The Redfish interface implementations need similar, but different, properties in the `driver_info` field, including `redfish_address`, `redfish_system_id`, `redfish_username`, and `redfish_password`. See the Redfish driver documentation.

Changing that TripleO command to set both the Redfish and WS-Man properties in a nodes `driver_info` field when its `driver` is `idrac` is beyond the scope of this specification. That will be addressed by a TripleO project blueprint.

- Define `idrac` hardware type support of IPMI interface implementations. That could be done as a follow-on to this.

Solution

This specification proposes to solve the problem it describes by changing the `idrac` hardware type. Since the Ironic Driver composition reform, we have been allowed to have one vendor driver with options configurable per node instead of many drivers for every vendor.[1] The reforms goals include[2]:

```
* Make vendors in charge of defining a set of supported interface
  implementations in priority order

* Allow vendors to guarantee that unsupported interface implementations will
  not be used with hardware types they define. This is done by having a
  hardware type list all interfaces it supports.
```

Implementing the solution in the `idrac` hardware type contributes toward making it the one Dell EMC driver for its bare metal servers with iDRACs and their value added implementations of the IPMI, Redfish, and WS-Man management protocols. It also aligns with the goals of the reform. That is what operators have come to expect.

Here are the details of the proposal.

- Define two new groups of interface implementations with entrypoints named `idrac-redfish` and `idrac-wsman`. The `idrac-redfish` entrypoints refer to Redfish interface implementations which are compatible with the iDRAC, presently those of the `management` and `power` hardware interfaces. The `idrac-wsman` entrypoints are new names for the legacy `idrac` entrypoints. The legacy `idrac` entrypoints are left unchanged. For example:

```
ironic.hardware.interfaces.management =
    ...
    idrac = ironic.drivers.modules.drac.management:DracManagement
    idrac-redfish = ironic.drivers.modules.drac.
↪management:DracRedfishManagement
    idrac-wsman = ironic.drivers.modules.drac.
↪management:DracWSManManagement
    ...
    redfish = ironic.drivers.modules.redfish.management:RedfishManagement
```

[1] See the *introduction* paragraph of the Ironic Driver composition reform.
[2] See the *Introduction* subsection in the *Proposed change* section of the Ironic Driver composition reform.

- Declare `idrac` hardware type support for the `idrac`, `idrac-redfish`, and `idrac-wsman` interface implementations. `idrac` continues to have the highest priority by being first in its `supported_<INTERFACE>_interfaces` lists. Here `<INTERFACE>` is a type of hardware interface: `inspect`, `management`, `power`, etc. For example:

```
class IDRACHardware(generic.GenericHardware):
    ...
    @property
    def supported_management_interfaces(self):
        return [management.DracManagement, management.DracWSManManagement,
            management.DracRedfishManagement]
    ...
```

Note

The property uses classes, not instances nor entrypoint names. The example assumes the required modules are imported.

- New `idrac-redfish` entrypoints are defined by new Python classes, because using the generic, vendor-independent Redfish classes would make the `redfish` entrypoints synonyms for `idrac-redfish` and supported. A later requirement to change the name of an entrypoints Python class to resolve a Dell EMC-specific incompatibility or introduce vendor value added, which would eliminate support for `redfish`, could be a breaking change. The new Python classes are derived from the generic, vendor-independent Redfish classes.
- New `idrac-wsman` entrypoints are defined by new Python classes. Those classes are created by renaming the classes for the legacy `idrac` entrypoints from `Drac<INTERFACE>` to `DracWSMan<INTERFACE>`. Here `<INTERFACE>` refers to a type of hardware interface: `Inspect`, `Management`, `Power`, etc.

  The legacy `Drac<INTERFACE>` classes are redefined by simply deriving them from the new `DracWSMan<INTERFACE>` classes. For example:

```
class DracManagement(DracWSManManagement):
    pass
```

  That makes the legacy `Drac<INTERFACE>` classes aliases for the new `DracWSMan<INTERFACE>` classes. Any bug fixes or features added to the WS-Man interface implementations are available from both the `idrac` and `idrac-wsman` entrypoints. Having separate classes for the two groups of entrypoints makes it possible to subsequently add logic that implements deprecation of the legacy `idrac` entrypoints by emitting a log message and similar.

Alternatives

- We could change the lowest layer of `python-dracclient` to support Redfish, in addition to WS-Man. However, we expect it would be challenging to provide `python-dracclient` APIs and workflows which abstract the very different Redfish and WS-Man technologies. Redfishs interface is RESTful, while WS-Man is a Simple Object Access Protocol (SOAP). APIs and workflows would likely need to be changed or newly defined. That would require substantial modification of the existing `idrac` interface implementations.
- We could maintain the status quo split of the `idrac` hardware type for WS-Man and `redfish` hardware type for Redfish. However, that would not promote and accelerate the use of Redfish among

operators with Dell EMC bare metal servers today, because `redfish` does not offer everything `idrac` does. That also would not support resolving Dell EMC vendor- specific incompatibilities with the generic, vendor-independent `redfish` hardware type nor using Redfish to introduce vendor value added.

- We could let the `redfish` interface implementations use Redfish OEM extensions to address vendor-specific incompatibilities and introduce vendor value added. However, that seems inconsistent with the intent that they be generic and vendor-independent.

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

ironic CLI

None

openstack baremetal CLI

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

None

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

- A deployer can add `idrac-redfish` to the `enabled_management_interfaces` and `enabled_power_interfaces` options to enable those new interface implementations.
- A deployer can add `idrac-wsman` to the `enabled_inspect_interfaces`, `enabled_management_interfaces`, `enabled_power_interfaces`, `enabled_raid_interfaces`, and `enabled_vendor_interfaces` to enable those new interface implementations.
- A deployer must specify properties in the nodes `driver_info` field that are needed by Redfish interface implementations, including `redfish_address`, `redfish_system_id`, `redfish_username`, and `redfish_password`, to use the `idrac-redfish` interface implementations. That is in addition to the legacy properties the `idrac` hardware type has needed in `driver_info` `drac_address`, `drac_username`, and `drac_password`.

```
openstack baremetal node create --driver idrac --driver-info \
  drac_address=1.2.3.4 --driver-info drac_username=admin --driver-info \
  drac_password=password --driver_info redfish_address=https://1.2.3.4 \
  --driver-info redfish_system_id=/redfish/v1/Systems/System.Embedded.1 \
  --driver-info redfish_username=admin --driver-info \
  redfish_password=password
```

  See the Redfish driver documentation, iDRAC driver documentation, and *Non-goals*.

- A deployer can specify the new `idrac-redfish` and `idrac-wsman` interface implementations on node enrollment:

```
openstack baremetal node create --driver idrac ... --management-interface␣
↪\
  idrac-wsman --power-interface idrac-wsman ...
```

  They can also be set by the following command:

```
openstack baremetal node set <NODE> --management-interface idrac-redfish \
  --power-interface idrac-redfish
```

  They must be enabled as described above.

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
rpioso

Other contributors:
None

Work Items

- Define two new groups of interface implementations with entrypoints named `idrac-redfish` and `idrac-wsman`.
- Declare `idrac` hardware type support for the `idrac`, `idrac-redfish`, and `idrac-wsman` interface implementations.
- Integration test the changes against Dell EMC bare metal servers.
- Modify the Dell EMC Ironic third-party continuous integration (CI) to cover supported configurations added by this specification.
- Update the iDRAC driver documentation.

Dependencies

This specification is related to the Driver composition reform.

It specifically targets Dell EMC bare metal servers equipped with an iDRAC and managed by the `idrac` hardware type.

Testing

This is not testable in the gate given current limitations on the availability of the specific hardware required.

The mitigation plan is to add coverage to the Dell EMC Ironic third-party CI for supported configurations added by this specification that we expect to be common.

Upgrades and Backwards Compatibility

This change is designed to be backwards compatible. The legacy `idrac` interface implementation entrypoints will be supported for at least some time. A separate story will cover their deprecation.

We will recommend switching to the appropriate new `idrac-redfish` and `idrac-wsman` interface implementation entrypoints as soon as it is possible.

Documentation Impact

The iDRAC driver documentation is updated to:

- describe switching from the legacy `idrac` interface implementation entrypoints to the new `idrac-redfish` and `idrac-wsman` entrypoints,
- reflect the changes to the supported interface implementations, and
- inform that a node configuration with a mix of Redfish and WS-Man `management` and `power` interface implementations is not supported.

References

OpenStack software projects:

- ironic
- python-dracclient
- sushy

Related Ironic specifications:

- Driver composition reform

Documentation:

- iDRAC driver
- Redfish driver
- TripleO register nodes

Standards:

- IPMI
- Redfish
- WS-Man

Out-of-band disk-erase for Gen10 and above HPE Proliant Servers

https://storyboard.openstack.org/#!/story/2004786

This specification proposes implementation of out-of-band disk-erase for iLO5 managed HPE Proliant servers.

Problem description

In the current scenario where disk-erase on HPE Proliant servers is done only via inband cleaning, iLO5 based HPE Proliant Gen10 servers provide support to perform out-of-band disk-erase which was not there in Gen9 and older servers. However, disk-erase request would be accepted by iLO only when system boot completes POST. Hence disk-erase needs to be accompanied by a reboot.

Proposed change

This spec proposes to implement out-of-band disk-erase clean_step in hardware type `ilo5` under new management interface `Ilo5Management` which would be inherited from existing management interface `IloManagement`.

List of changes required:

- The following would be the composition of the new management interface `Ilo5Management`:
  - `erase_devices` - This will erase all disks on the baremetal node.
    - *erase_devices* will call proliantutils library method *do_disk_erase* to perform the operation in iLO. User can also choose between different erase pattern (ex. block, overwrite, crypto, zero) to perform the disk erase operation.
    - The reboot is required to initiate the disk erase. The actual disk erase operation would take time based on disk type and size.

Alternatives

One can perform in-band disk-erase to achieve the same result. However, The ramdisk to be used in such case should have proliant-tools element that bundles ssacli utility required for disk-erase operations as part of the image.

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

None

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

Users need to configure two options to make use of OOB disk-erase on HPE Proliant Gen10 servers.

- Configure the hardware type `ilo5` to ([DEFAULT] `enabled_hardware_types`).
- Configure the new management interface `ilo5` to ([DEFAULT] `enabled_management_interfaces`).

Developer impact

None

Implementation

Assignee(s)

Primary assignee: pareshsao

Work Items

- Add a new management interface `Ilo5Management` to hardware type `ilo5`
- Writing unit-test cases for the new OOB disk-erase interface.

Dependencies

Support for OOB disk-erase in proliantutils is under development and is yet to be released.

Testing

Unit test cases will be added. Will be tested in 3rd party CI setup.

Upgrades and Backwards Compatibility

None

Documentation Impact

Need to update iLO driver documentation for new management interface.

References

None

Support for Software RAID

https://storyboard.openstack.org/#!/story/2004581

This spec proposes to add support for the configuration of software RAIDs.

In analogy to the way hardware RAIDs are currently set up, the RAID setup shall be done as part of the cleaning (clean-time software RAID). Admin Users define the target RAID config which will be applied whenever the node is cleaned, i.e. before it becomes available for instance creation.

In order to allow the End User to provide details on how the software RAID shall be configured, the RAID setup should eventually become part of the deployment steps. Integrating this into the deployment steps framework, however, is beyond the scope of this spec.

Problem description

As it is hardware agnostic, flexible, reliable, and easy to use, software RAID has become a popular choice to protect against disk device failures - also in production setups. Large deployments, such as the ones at Oath or CERN, rely on software RAID for their various services.

Ironics current lack of support for such setups requires Deployers and Admins to withdraw to workarounds in order to provide their End Users with physical instances based on a software RAID configuration. These workarounds may require to maintain an additional installation infrastructure which is then either integrated into the installation process or requires the End User to re-install a machine a second time after it has been already provisioned by Ironic to eventually end up with the desired configuration of the disk devices. This increases the complexity for Deployers and Admins, and can also lead to a decrease of the End Users satisfaction with the overall provisioning and installation process.

Proposed change

The proposal is to extend Ironic to support software RAID by:

- using a nodes `target_raid_config` to specify the desired s/w RAID layout (with some restrictions, see below);
- adding support in the `ironic-python-agent` to understand a software RAID config as specified in a nodes `target_raid_config` and be able to create and delete such configurations;
- allow the `ironic-python-agent` to consider s/w RAID devices for deployment, e.g. via root device hints (considering them at all is already addressed in [1]);
- adding support in Ironic and the `ironic-python-agent` to take the necessary steps to boot from a s/w RAID, e.g. installing the boot loader on the correct device(s).

Initially, only the following configurations will be supported for the `target_raid_config` as to be set by the Admin:

- a single RAID-1 spanning the available devices and serving as the deploy target device, or
- a RAID-1 serving as the deploy target device plus a RAID-N where the RAID level N is configurable by the Admin. N can be 0, 1, 5, 6, or 10.

The supported configurations have been limited to these two options in order to avoid issues when booting from RAID devices. Having a (small) RAID-1 device to boot from is a common approach when setting up more advanced RAID configurations: a RAID-1 holder device can look like a standalone disk and does not require the bootloader to have any knowledge or capabilities to understand more complex RAID configurations.

In order to signal that a software RAID configuration is indeed desired (and to protect from a situation where a software RAID is set up accidentally when the configuration passed via the `target_raid_config` was meant for a hardware RAID setup, for instance), the `controller` property of all of the logical disks needs to be set to `software`. Without this setting, the software RAID code in the GenericHardwareManager of the IPA will ignore the given `target_raid_config`. If it is set on only one of the logical drives, the validation code will raise an error.

The `controller` property set to `software` will also be used by the conductor to identify a software RAID and trigger the required installation of the bootloader. While whole-disk images are expected to come with a bootloader configuration as part of the image, for software RAIDs in the current design the image will not be at the start of a real disk, but inside the first partition on top of a software RAID-1. The bootloader must hence be explicitly installed onto the underlying holder disks, and this property will indicate when to do this.

An example of a valid software RAID configuration would hence look like:

```
{
    "logical_disks": [
        {
            "size_gb": 100,
            "raid_level": "1",
            "controller": "software"
        },
        {
            "size_gb": "MAX",
            "raid_level": "0",
            "controller": "software"
        }
    ]
}
```

Support for more than one RAID-N, support for the selection of a subset of drives to act as holder devices, support for simultaneous software and hardware RAID devices as well as support to partition the created RAID-N device are left for follow-up enhancements and beyond the scope of this specification.

Also, there is currently no support for partition images, only whole disk images are supported.

A first prototype very close to the proposal is available from [2][3][4].

Alternatives

As mentioned above, the alternative is to use other methods to create s/w RAID setups on physical nodes and integrate these out-of-band approaches into the provisioning workflow of individual deployments. This increases complexity on the Deployer/Admin side and can have a negative impact on the user experience when creating physical instances which need to have a software RAID setup..

Data model impact

None.

State Machine Impact

None.

REST API impact

None.

Client (CLI) impact

None.

ironic CLI

None.

openstack baremetal CLI

None.

RPC API impact

None.

Driver API impact

The proposed functionality could be consolidated into a new RAID interface.

Nova driver impact

None.

Ramdisk impact

The `ironic-python-agent` will need to be able to: * setup and clean software RAID devices * consider software RAID devices for deployment * configure the holder devices of the RAID-1 device in a way they are bootable

This functionality could be consolidated in an additional RAID interface.

Security impact

None.

Other end user impact

While the predefined RAID-1 ensures that a system should be able to boot, End Users need to be aware that the kernel of the started image needs to be able to understand software RAID devices.

Scalability impact

None.

Performance Impact

None.

Other deployer impact

Deployers will need to be aware that the configuration and clean up of the RAID-N devices is only done during cleaning, so any changes require the node to be cleaned. Also, the config is not configurable by the End User, but limited to admins (as the target_raid_config) is a node property. All of this, however, already holds true for hardware RAID configurations.

Developer impact

None.

Implementation

An initial proof-of-concept is available from [2][3][4].

Assignee(s)

Primary assignee:
None.

Other contributors:
Arne.Wiebalck@cern.ch (arne_wiebalck)

Work Items

This is to be defined once the overall idea is accepted and theres agreement on a design.

Dependencies

None.

Testing

TBD

Upgrades and Backwards Compatibility

None.

Documentation Impact

Documentation on how to configure a software RAID along with the limitations outlined in Deployers Impact need to be documented.

References

[1] https://review.opendev.org/#/c/592639 [2] CERN Hardware Manager: https://github.com/cernops/cern-ironic-hardware-manager/commit/7f6d892ec4848a09000ed1f28f3137bf8ba917f0 [3] Patched Ironic Python Agent: https://github.com/cernops/ironic-python-agent/commit/bddac76c4d100af0103a6bc08b81dd71681a9c02 [4] Patched Ironic: https://github.com/cernops/ironic/commit/581e65f1d8986ac3e859678cb9aadd5a5b06ba60

Add Intelő Speed Select Support

https://storyboard.openstack.org/#!/story/2005390

Multiple types of servers are needed to handle diverse workloads. Purchasing and managing these servers introduces complexity and increases total cost of ownership(TCO). Intel Speed Select Technology(SST)[1] is a collection of features that improves performance and optimizes TCO by providing more control over CPU performance. With Intel SST, one server can do more which means the same server can be used to run different workloads. One of the feature is Intel Speed Select Technology-Performance Profile (SST-PP) which allows configuring the CPU to run at 3 distinct operating points or profiles.

With Intel SST-PP, one can set the desired core count and their base and turbo frequencies which can help to tune the server to specific workload performance.

Problem description

This spec proposes to support Intel SST-PP feature in Ironic. With Intel SST-PP, Ironic users can:

- Run their servers at different configuration level.
- Each configuration level supports different number of active cores and frequency.
- Same server can be used to run multiple different workloads thus decreasing TCO.

Intel SST-PP Speed Select supports three configuration levels:

- 0 - Intel SST-PP Base Config
- 1 - Intel SST-PP Config 1
- 2 - Intel SST-PP Config 2

Following table shows the list of active cores and their base frequency at different SST-PP config levels:

```
-----------------------------------------
|   Config    |  Cores  |  Base Freq (GHz)  |
-----------------------------------------
| Base        | 24      | 2.4               |
| Config 1    | 20      | 2.5               |
```

(continues on next page)

(continued from previous page)

```
| Config 2     | 16       | 2.7                |
-----------------------------------------------
```

Proposed change

Intel SST-PP can be set over IPMI. Each configuration level has its own hexa raw code that the server understands. Ironic sends this code to the server via IPMI to set the desired SST-PP level.

We will map these configurations to traits that Ironic understands.

- 0 - `CUSTOM_INTEL_SPEED_SELECT_CONFIG_BASE`
- 1 - `CUSTOM_INTEL_SPEED_SELECT_CONFIG_1`
- 2 - `CUSTOM_INTEL_SPEED_SELECT_CONFIG_2`

The solution works at two stages:

Scheduling

The solution to support Intel SST-PP is first enable scheduling the request for baremetal instance deployment on the nodes that supports it. We set the desired configuration as the trait in our flavor:

```
$ openstack flavor set --property \
  trait:CUSTOM_INTEL_SPEED_SELECT_CONFIG_2=required baremetal
```

Now, we also need to update the Ironic nodes trait with the supported configuration levels:

```
$ openstack baremetal node add trait node-0 \
  CUSTOM_INTEL_SPEED_SELECT_CONFIG_BASE CUSTOM_INTEL_SPEED_SELECT_CONFIG_1 \
  CUSTOM_INTEL_SPEED_SELECT_CONFIG_2
```

Now, when user sends a request to boot a node with the `baremetal` flavor, placement API service will select the Ironic node that supports Intel SST-PP.

Provisioning

The Intel SST-PP needs to be set via IPMI before powering on the node in the process of provisioning. Ironic API service receives the desired configuration in `node.instance_info.traits` in the boot request. Ironic will then run the deploy templates step matching the trait. The deploy template will specify the new `configure_intel_speedselect` step which configures the Intel SST-PP configuration level and then powers on the node.

Ironic will need below details to configure Intel SST-PP on the node:

- Intel SST-PP configuration level: This is the required configuration level on the node. Possible values are the traits listed above. This information is set by admins in the flavors trait they want to boot the baremetal node with. Nova in turn updates the Ironics node information with the trait.
- Number of sockets: This is the number of sockets per CPU. Setting Intel SST-PP needs to be done for every socket.

Both these information can be provided as an argument to the `configure_intel_speedselect` deploy step.

Alternatives

Continue to not support Intel SST-PP or users can manually configure it independent of Ironic.

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

ironic CLI

None

openstack baremetal CLI

None

RPC API impact

None

Driver API impact

- Add a new hardware type `IntelIPMIHardware` to support Intel SST-PP enabled servers.

```
class IntelIPMIHardware(IPMIHardware):
```

- Add a new management interface `IntelIPMIManagement` to manage the configuration of Intel SST-PP on the servers. This class will have a new deploy step `configure_intel_speedselect` to configure Intel SST-PP on the nodes. This will also be enabled as a clean step so that it can be used to reset the configuration while node cleaning.

```
class IntelIPMIManagement(ipmitool.IPMIManagement):
    @base.deploy_step(priority=200, argsinfo={
        'intel_speedselect_config': {
            'description': (
                 "Intel SST-PP configuration."
            ),
            'required': True
        },
        'socket_count': {
            'description': (
                "No. of sockets."
```

(continues on next page)

(continued from previous page)

```
            )
        }
    })
    def configure_intel_speedselect(self, task, **kwargs):
        return None
```

The following table contains the `intel_speedselect_config` values for the proposed deploy templates:

```
-------------------------------------------------------------------------
|   Deploy Template                     |  intel_speedselect_config   |
-------------------------------------------------------------------------
| CUSTOM_INTEL_SPEED_SELECT_CONFIG_BASE  |  0x00                       |
| CUSTOM_INTEL_SPEED_SELECT_CONFIG_1     |  0x01                       |
| CUSTOM_INTEL_SPEED_SELECT_CONFIG_2     |  0x02                       |
-------------------------------------------------------------------------
```

Nova driver impact

None

Ramdisk impact

None

Security impact

None

Other end user impact

Users will have to update the traits with the desired Intel SST-PP configuration level in the corresponding flavors.

Scalability impact

None

Performance Impact

None

Other deployer impact

Deployers wishing to use feature will have to add the Intel SST-PP configuration in Nodes trait and also have to create corresponding deploy templates.

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
mkrai

Work Items

- Implement Intel SST-PP support code in Ironic.
- Write the test code.
- Write a document explaining how to use Intel SST-PP.

Dependencies

None

Testing

- Unit tests

Upgrades and Backwards Compatibility

None

Documentation Impact

New document will be added to explain the support of new Intel SST-PP feature in Ironic and how to use it.

References

[1] https://www.intel.com/content/www/us/en/architecture-and-technology/speed-select-technology-article.html

5.6 Stein

5.6.1 12.1

Allocation API

https://storyboard.openstack.org/#!/story/2004341

This spec proposed creating of API for *allocation* of nodes for deployment.

Problem description

The users of standalone ironic do not have an out-of-box means to find a suitable node to deploy onto. The metalsmith project was created to add this gap short-term, but it is not suitable for consumer code that is not written in Python. A potential consumer is a K8S provider for standalone ironic.

The API user story is as follows:

> Given a resource class and, optionally, a list of required traits, return me an available bare metal node and set `instance_uuid` on it to make it as reserved.

Proposed change

Overview

This RFE proposed a new ReST API endpoint `/v1/allocations` that will initially allow creating and deleting *Allocation* resources.

Two implementations of the allocation process are planned:

1. Via the database, similar to how metalsmith now operates.
2. Via the Placement service, similar to how nova currently operates.

This spec concentrates on the API design and the first (standalone) case.

Allocation process

An allocation happens as follows:

1. An API client sends a `POST /v1/allocations` request, specifying a resource class, and optionally traits and node UUID.
2. The allocation request is routed to a random available conductor.
3. The conductor creates an *Allocation* object in the database with `state=allocating` and `conductor_affinity=<host name>`.
4. A thread is spawned for the remaining allocation process, and the allocation object is returned to the caller.

Allocation: database backend

The following actions are done by the conductor handling the allocation when database is used as backend (the only option in this spec):

1. Fetch list of nodes from the database with:
   - `provision_state=available`
   - `maintenance=False`
   - `power_state!=None`

   Note

   This is required for compatibility with really old API versions that allow creating nodes directly in the `available` state.

- `instance_uuid=None`
- `resource_class=<requested resource class>`
- `uuid` in the list of candidate nodes (if provided)
- requested traits are a superset of node traits

2. If the list is empty, set allocations `state` to `error` and `last_error` to the explanation.
3. Shuffle the list, so that several processes do not try reserving nodes in the same order.
4. Acquire a lock on the first node. If locking succeeds, check that the assumptions are still valid about this node, and reserve it by setting its `instance_uuid` to the `uuid` of the allocation. In the same database transaction:
   - set allocations `node_id` to the nodes ID,
   - set allocations `state` to `active`,
   - set nodes `allocation_id` to the allocations ID,
   - add matched traits to nodes `instance_info`.

   Note

   Since the conductor may not have the hardware type for the selected node, we will update TaskManager to avoid constructing the `driver` object.

5. If something fails on the previous step, proceed to the next node. If no more nodes are left, set the allocations `state` to `error` and `last_error` to the explanation.

Deallocation: database backend

The deallocation process will in one transaction:

- unset nodes `instance_uuid`,
- unset nodes `allocation_id`,
- delete the allocation.

The deallocation is triggered either explicitly via API or when a node is torn down (at the same time when nodes `instance_uuid` and `instance_info` are purged).

Note

In the future we might consider supporting *sticky allocations* which survive nodes tear down. This is outside the scope of this spec.

There is one important difference between using just `instance_uuid` and using the allocation API: `instance_uuid` can be set and unset for `active` nodes, while for allocations it will be forbidden. The reason is that with the future Placement backend removing an allocation would mark the node as available in Placement.

HA and take over

- When a conductor restarts, it fetches allocations with
  - `conductor_affinity=<host name>`
  - `state=allocating`

  and starts the allocation procedure for each of them.
- If the conductor handling an allocation stops without a replacement, the reservation becomes orphaned. All conductors periodically fetch list of allocations belonging to dead conductors and each tries to resume them.

  First, it tries to update the `conductor_affinity` by doing something like:

```
UPDATE allocations SET conductor_affinity=<new host name>
   WHERE id=<allocation ID> AND conductor_affinity=<dead host name>
```

  If the query updated 1 row, we know that the new conductor now manages the allocation. Otherwise we know that another conductor took it over.
- To avoid rare races with this take over procedure, the normal update will also look like:

```
UPDATE allocations SET <new values>
    WHERE id=<allocation ID> AND conductor_affinity=<current host name>
```

Alternatives

- Make each consumer invent their own allocation procedure or use metalsmith.
- Write a new service for managing reservations (probably based on metalsmith code base).
- Make the API blocking and avoid having states for allocations. Such an approach would result in easier API and implementation, but it can be problematic when using an external system, such as Placement, as a backend, since the requests to it make block the RPC thread.

  Additionally, the asynchronous design will make it easier to introduce a bulk allocation API in the future, if we decide so.

Data model impact

Introduce a new database/RPC object *Allocation* with the following fields:

- `id` internal integer ID, not exposed to users.
- `uuid` unique UUID of the allocation.
- `name` unique name of the allocation, follows the same format as nodes names.

  Note

  This field is useful, for example, for systems using host names, like metalsmith.
- `node_id` reserved node ID (can be `null`) - foreign key to the `nodes` table.
- `updated_at/created_at` standard update/creation date time fields.

- `resource_class` mandatory requested resource class.
- `candidate_nodes` list of node UUIDs to choose from (can be `null`).

> **Note**
>
> This allows a caller to pre-filter nodes by arbitrary criteria.

- `state` allocation state, see *State Machine Impact*.
- `last_error` last error message.
- `conductor_affinity` internal field, specifying which conductor currently handles this allocation.

Introduce a helper table `allocation_traits` mapping an allocation to its requested traits (very similar to `node_traits`).

Update the `nodes` table with a new foreign key `allocation_id`. It will be set to a ID of a corresponding allocation. Unlike `instance_uuid`, it will only be set when an allocation is created. If `allocation_id` is not empty, `instance_uuid` will hold the UUID of the corresponding allocation (the opposite is not necessary true).

State Machine Impact

No impact on the node state machine.

This RFE introduces a simple state machine for *Allocation* objects, consisting of three states:

- `allocating` allocation is in progress (initial state).
- `active` allocation active.
- `error` allocation failed.

In the initial version only the following paths are possible:

- from `allocating` to `active` on success.
- from `allocating` to `error` on failure.

In the future we may allow moving from `error` back to `allocating` to retry the allocation.

REST API impact

- `POST /v1/allocations` create an allocation.

  The API accepts a JSON object. The following field is mandatory:

  - `resource_class` requested nodes resource class.

  The following fields are accepted:

  - `uuid` to create an allocation with the specific UUID.
  - `candidate_nodes` to limit the query to one of these nodes.

> **Note**
>
> This value is converted from names to UUIDs internally.

  - `traits` list of requested traits.
  - `name` allocation name.

  The normal response is HTTP CREATED with the response body being the created allocation representation. An allocation is created in the `allocating` state.

  Error codes:

  - 400 Bad Request if
    - any node from `candidate_nodes` cannot be found (by name or UUID).
    - the `resource_class` value is invalid.
    - `traits` is provided and is not a list of valid trait strings.
    - `name` is not an accepted name.
  - 406 Conflict if
    - the provided `uuid` is not unique or matches `instance_uuid` of any node.
    - the provided `name` is not unique.

- `GET /v1/allocations` list allocations.

  Parameters:

  - `fields` list of fields to retrieve for each allocation.
  - `state` filter allocations by the state.
  - `resource_class` filter allocations by resource class.
  - `node` filter allocations by node UUID or name.

  Error codes:

  - 400 Bad Request if
    - `state` is invalid.
    - `resource_class` is invalid.
    - `node` does not exist.
    - any of the requested fields is invalid.

- `GET /v1/allocations/<uuid or name>` retrieve an allocation.

  Parameters:

  - `fields` list of fields to retrieve.

  Error codes:

  - 400 Bad Request if any of the requested fields is invalid.
  - 404 Not Found if the allocation is not found.

- `DELETE /v1/allocations/<uuid or name>` remove the allocation and release the node.

  No request or response body. Response code is HTTP 204 No Content.

  Error codes:

  - 404 Not Found if the allocation is not found.
  - 409 Conflict if the corresponding node is `active` or is in a state where updates are not allowed.

  Note

  This request will succeed only for real allocations. It will not be possible to unset `instance_uuid` created without an allocation (i.e. by direct `PATCH` to a node) using this API.

- `GET /v1/nodes/<node UUID or name>/allocation` get allocation associated with the node.

  Parameters:

  - `fields` list of fields to retrieve.

  The response body is the *Allocation* object representation.

  Error codes:

  - 404 Not Found if
    - the node cannot be found.
    - there is no allocation for the node.
  - 400 Bad Request if
    - node has `instance_uuid` that does not correspond to any allocation.
    - any of the requested fields is invalid.

- Update `GET /v1/nodes`, `GET /v1/nodes/detail` and `GET /v1/nodes/<node UUID or name>`:

  Expose the new `allocation_uuid` field (converted from the nodes `allocation_id`).

- Update `PATCH /v1/nodes/<node UUID or name>`:

  If `instance_uuid` is unset and the current value corresponds to an allocation:

  - if node is `active` or in a state that disallows updates, and `maintenance` mode is off, return HTTP 409 Conflict,
  - otherwise delete the allocation.

  If `instance_uuid` is set, do NOT create an allocation, keep the previous behavior.

  Note

  This is needed to avoid affecting the nova virt driver. This decision may be revisited in future API versions.

The `allocation_uuid` field is read-only, an attempt to change it directly will result in HTTP 400 (Bad Request).

- Update `DELETE /v1/nodes/<node UUID or name>`:

  If a node is deleted with allocation (possible only in maintenance mode), the allocation is deleted as well.

Client (CLI) impact

ironic CLI

None.

openstack baremetal CLI

The matching commands will be created:

```
openstack baremetal allocation create --resource-class <class> \
   [--trait <trait>] [--trait <trait>] [--uuid <uuid>] [--name <name>]
openstack baremetal allocation list [--state <state>] [--fields <fields>]
   [--resource-class <class>] [--node <UUID or name>]
openstack baremetal allocation get <uuid or name> [--fields <fields>]
openstack baremetal allocation delete <uuid or name>
```

The `allocation_uuid` field will be exposed.

RPC API impact

Two new RPC calls are introduced:

- Creating an allocation

```
def create_allocation(self, context, allocation):
   """Create an allocation.

   Creates the provided allocation in the database, then starts a thread
   to process it.

   :param context: context
   :param allocation: allocation object.
   """
```

- Deleting an allocation

```
def destroy_allocation(self, context, allocation):
   """Destroy an allocation.

   Removes the allocation from the database and releases the node.

   :param context: context
   :param allocation: allocation object.
   """
```

metalsmith impact

The metalsmith project implements a superset of the proposed feature on a client side. After this API is introduced, metalsmith will switch the `reserve_node` call to using it in the following way:

- If the request contains a `resource_class` and, optionally, `traits` and candidate nodes, the new API will be used.
- If the request contains anything not supported by the new API, metalsmith will continue client-side node filtering, and will create an allocation with a list of suitable nodes.

Driver API impact

None

Nova driver impact

None

In the future we may use the allocation API in the nova driver, but there are no plans for it now. Currently going through the allocation API will result in an attempt of double allocation in Placement if Placement is used as an allocation backend.

Ramdisk impact

None

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

A new periodic task will run on each conductor to periodically check for stalled reservations belonging to dead conductors. The default period will be 60 seconds. It will be possible to disable it, in which case the allocations may get stuck forever if their assigned conductor dies.

Other deployer impact

None

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
dtantsur

Work Items

- Add new tables and the *Allocation* RPC object.
- Add RPC for allocating/deallocating.
- Add API for allocations creation and deletion, and API reference.
- Update conductor starting procedure to check for unfinished allocations.
- Add a periodic task to check for orphaned unfinished allocations.

Dependencies

None

Testing

- Unit tests and Tempest API will be provided.
- The standalone integration tests will be updated to use the new API.
- We can add support for the new API to bifrost (e.g. via metalsmith), and test it in a bifrost CI job.

Upgrades and Backwards Compatibility

This change is fully backward compatible. Code using `instance_uuid` for allocations is not affected.

Documentation Impact

API reference will be provided.

References

Deploy Templates

https://storyboard.openstack.org/#!/story/1722275

This builds on the deployment steps spec, such that it is possible to request a particular set of deployment steps by requesting a set of traits.

When Ironic is used with Nova, this allows the end user to pick a Flavor, that requires a specific set of traits, that Ironic turns into a request for a specific set of deployment steps in Ironic.

Problem description

One node can be configured in many ways. Different user workloads require the node to be configured in different ways. As such, operators would like a single pool of hardware to be reconfigured to match each users requirements, rather than attempting to guess how many of each configurations are required in advance.

In this spec, we are focusing on reconfiguring regular hardware to match the needs of each users workload. We are not considering composable hardware.

When creating a Nova instance with Ironic as the backend, there is currently no way to influence the deployment steps used by Ironic to provision the node.

Example Use Case

Consider a bare metal node that has three disks. Different workloads may require different RAID configurations. Operators want to test their hardware and determine the specific set of RAID configurations that work best, and offer that choice to users of Nova, so they can pick the configuration that best matches their workload.

Proposed change

Context: Deployment Steps Framework

This spec depends on the deployment steps framework spec. The deployment steps framework provides the concept of a deployment step that may be executed when a node is deployed.

It is assumed that there is a default set of steps that an operator configures to happen by default on every deploy. Each step task has a priority defined, to determine the order that that step runs relative to other enabled deploy steps. Configuration options and hard-coded priorities define if a task runs by default or not. The priority says when a task runs if the task should run during deployment.

Deploy Templates

This spec introduces the concept of a `Deploy Template`. It is a mapping from a valid trait name for the node to one or more Deployment Steps and all the arguments that should be given to those deployment steps. There is a new API added to Ironic for Operators to specify these `Deploy Templates`.

To allow a `Deploy Template` for a given Ironic node, you add the trait name of the deploy template to the `traits` list on each Ironic node that needs the deploy template enabled. The validation of the node must fail if any of the enabled `Deploy Templates` are not supported on that node.

It is worth noting that all traits set on the node are synchronised to Novas placement API. In turn, should a user request a flavor that requires a trait (that may or may not map to a deploy template) only nodes that have the trait set will be offered as candidates by Novas Placement API.

To request the specified `Deploy Template` when you provision a particular Ironic node, the corresponding trait is added to a list in `node.instance_info` under the key of `traits`. This is what Novas ironic virt driver already does for any required trait in the flavors extra specs for that instance. Again, Ironic already validates that the traits in `node.instance_info` are a subset of the traits that the Operator has set on the Ironic node, via the new node-traits API.

During the provisioning process Ironic checks the list of traits set in `node.instance_info` and checks if they match any `Deploy Templates`. The list of matching `Deploy Templates` are then used to extend the list of deployment steps for this particular provision operation. As already defined in the deployment

steps framework, this is then combined with the list of deployment steps from what is configured to happen by default for all builds.

The order in which the steps are executed will be defined by the priority of each step. If the code for a deploy step defines a deploy priority of 0, and that is not changed by a configuration option, that deploy step does not get executed by default. If a deploy template specifies a priority (this is required if the code has a default priority of 0), this overrides both the code default and any configuration override.

It is acceptable to request the same deploy step more than once. This could be done to execute a deploy step multiple times with different arguments, for example to partition multiple disks.

Any deploy step in a requested deploy template will override the default arguments and priority for that deploy step. 0 is the only priority override that can be set for any of the core deploy steps, i.e. you can only disable the core step, you cant change the order of its execution.

Trait names used in Deploy Templates should be unique - no two deploy templates should specify the same trait name.

In summary, you can use traits to specify additional deploy steps, by the mapping between traits and deploy steps specified in the new `Deploy Templates` API.

Current Limitations

When mapping this back to Nova integration, currently there would need to be a flavor for each of these combinations of traits and resource_class. Longer term Nova is expected to offer the option of a user specifying an override trait on a boot request, based on what the flavor says is possible. This spec has no impact on the Nova ironic virt driver beyond what is already implemented to support node-traits.

Example

Lets now look at how we could request a particular deploy template via Nova.

`FlavorVMXMirror` has these extra specs:

- `resource:CUSTOM_COMPUTE_A = 1`
- `trait:CUSTOM_CLASS_A = required`
- `trait:CUSTOM_BM_CONFIG_BIOS_VMX_ON = required`
- `trait:CUSTOM_BM_CONFIG_RAID_DISK_MIRROR = required`

`FlavorNoVMXStripe` has these extra specs:

- `resource:CUSTOM_COMPUTE_A = 1`
- `trait:CUSTOM_CLASS_A = required`
- `trait:CUSTOM_BM_CONFIG_BIOS_VMX_OFF = required`
- `trait:CUSTOM_BM_CONFIG_RAID_DISK_STRIPE = required`

Its possible the operator has set all of the Ironic nodes with `COMPUTE_A` as the resource class to have all of these traits assigned:

- `CUSTOM_BM_CONFIG_BIOS_VMX_ON`
- `CUSTOM_BM_CONFIG_BIOS_VMX_OFF`
- `CUSTOM_OTHER_TRAIT_I_AM_USUALLY_IGNORED`

- CUSTOM_BM_CONFIG_RAID_DISK_MIRROR
- CUSTOM_BM_CONFIG_RAID_DISK_STRIPE

The Operator has also defined the following deploy templates:

```
{
  "deploy-templates": [
    {
      "name": "CUSTOM_BM_CONFIG_RAID_DISK_MIRROR",
      "steps": [
        {
          "interface": "raid",
          "step": "create_configuration",
          "args": {
            "logical_disks": [
              {
                "size_gb": "MAX",
                "raid_level": "1",
                "is_root_volume": true
              }
            ],
            "delete_configuration": true
          },
          "priority": 10
        }
      ]
    },
    {
      "name": "CUSTOM_BM_CONFIG_RAID_DISK_STRIPE",
      "steps": [
        {
          "interface": "raid",
          "step": "create_configuration",
          "args": {
            "logical_disks": [
              {
                "size_gb": "MAX",
                "raid_level": "0",
                "is_root_volume": true
              }
            ],
            "delete_configuration": true
          },
          "priority": 10
        }
      ]
    },
    {
      "name": "CUSTOM_BM_CONFIG_BIOS_VMX_ON",
      "steps": [...]
    },
```

(continues on next page)

(continued from previous page)

```
    {
      "name": "CUSTOM_BM_CONFIG_BIOS_VMX_OFF",
      "steps": [...]
    }
  ]
}
```

When a Nova instance is created with `FlavorVMXMirror`, the required traits for that flavor are set on `node.instance_info['traits']` such that Ironic adds the deploy steps defined in `CUSTOM_BM_CONFIG_BIOS_VMX_ON` and `CUSTOM_BM_CONFIG_RAID_DISK_MIRROR`, and the node is appropriately configured for workloads that want that specific flavor.

Alternatives

Alternative approach

This design solves two problems:

1. I want to request some custom configuration to be applied to my bare metal server during provisioning.
2. Ensure that my instance is scheduled to a bare metal node that supports the requested configuration.

As with capabilities, the proposed design uses a single field (traits) to encode configuration and scheduling information. An alternative approach could separate these two concerns.

Deploy templates could be requested by a name (not necessarily a trait) or UUID as a nova flavor `extra_spec`, and pushed to a `deploy_templates` field in the ironic nodes `instance_info` field by the nova virt driver. Ironic would then apply the requested deploy templates during provisioning.

If some influence in the scheduling process is required, this could be provided by traits, but this would be a separate concern.

Adapting the earlier example:

`FlavorVMXMirror` has these extra specs:

- `resource:CUSTOM_COMPUTE_A = 1`
- `trait:CUSTOM_BM_CONFIG_BIOS_VMX_ON = required`
- `trait:CUSTOM_BM_CONFIG_RAID_DISK_MIRROR = required`
- `deploy_template:BIOS_VMX_ON=<?>`
- `deploy_template:BIOS_RAID_DISK_MIRROR=<?>`

Only ironic nodes supporting the `CUSTOM_BM_CONFIG_BIOS_VMX_ON` and `CUSTOM_BM_CONFIG_RAID_DISK_MIRROR` traits would be scheduled to, and the nova virt driver would set `instance_info.deploy_templates` to `BIOS_VMX_ON,BIOS_RAID_DISK_MIRROR`.

There are some benefits to this alternative approach:

- It would automatically support cases beyond the simple one trait mapping to one deploy template case we have here. For example, to support deploy template `X`, features `Y` and `Z` must be supported by the node (without combinatorial trait explosions).

- In isolation, the configuration mechanism is conceptually simpler - the flavor specifies a deploy template directly.
- It would work in standalone ironic installs without introducing concepts from placement.
- We dont overload the concept of traits for carrying configuration information.

There are also some drawbacks:

- Additional complexity for users and operators that now need to apply both traits and deploy templates to flavors.
- Less familiar for users of capabilities.
- Having flavors that specify resources, traits and deploy templates in `extra_specs` could leave operators and users scratching their heads.

Extensions

This spec attempts to specify the minimum viable feature that builds on top of the deployment steps framework specification. As such, there are many possible extensions to this concept that are not being included:

- While you can use standard traits as names of the deploy templates, it is likely that many operators will be forced into using custom traits for most of their deploy templates. We could better support the users of standard traits if we added a list of traits associated with each deploy template, in addition to the trait based name. This list of traits will act as an alias for the name of the deploy template, but this alias may also be used by many other deploy templates. The node validate will fail if for any individual node one of traits set maps to multiple deploy templates. To disambiguate which deploy template is requested, you can look at what deploy template names are in the chosen nodes trait list. For each deploy template you look at any other traits that can be used to trigger that template, eventually building up a trait to deploy template mapping for each trait set on the node (some traits will not map to any deploy template). That can be used to detect if any of the traits on the node map to multiple deploy templates, causing the node validate to fail.
- For some operators, they will end up creating a crazy number of flavors to cover all the possible combinations of hardware they want to offer. It is hoped Nova will eventually allow operators to have flavors that list possible traits, and a default set of traits, such that end users can request the specific set of traits they require in addition to the chosen flavor.
- While ironic inspector can be used to ensure each node is given an appropriate set of traits, it feels error prone to add so many traits to each Ironic node. It is hoped when a concept of node groups is added, traits could be applied to a group of nodes instead of only applying traits to individual nodes (possibly in a similar way to host aggregates in Nova). One suggestion was to use the Resource Class as a possible grouping, but that is only a very small part of the more general issue of groups nodes to physical networks, routed network segments, power distribution groups, all mapping to different ironic conductors, etc.
- There were discussions about automatically detecting which Deploy Templates each of the nodes supported. However most operators will want to control what is available to only the things they wish to support.

Data model impact

Two new database tables will be added for deploy templates:

```
CREATE TABLE deploy_templates (
    id INT(11) NOT NULL AUTO_INCREMENT,
    name VARCHAR(255) CHARACTER SET utf8 NOT NULL,
    uuid varchar(36) DEFAULT NULL,
    PRIMARY KEY (id),
    UNIQUE KEY `uniq_deploy_templaes0uuid` (`uuid`),
    UNIQUE KEY `uniq_deploy_templaes0name` (`name`),
)

CREATE TABLE deploy_template_steps (
    deploy_template_id INT(11) NOT NULL,
    interface VARCHAR(255) NOT NULL,
    step VARCHAR(255) NOT NULL,
    args TEXT NOT NULL,
    priority INT NOT NULL,
    KEY `deploy_template_id` (`deploy_template_id`),
    KEY `deploy_template_steps_interface_idx` (`interface`),
    KEY `deploy_template_steps_step_idx` (`step`),
    CONSTRAINT `deploy_template_steps_ibfk_1` FOREIGN KEY (`deploy_template_
↪id`) REFERENCES `deploy_templates` (`id`),
)
```

The `deploy_template_steps.args` column is a JSON-encoded object of step arguments, `JsonEncodedDict`.

New `ironic.objects.deploy_template.DeployTemplate` and `ironic.objects.deploy_template_step.DeployTemplateStep` objects will be added to the object model. The deploy template object will provide support for looking up a list of deploy templates that match any of a list of trait names.

State Machine Impact

No impact beyond that already specified in the deploy steps specification.

REST API impact

A new REST API endpoint will be added for deploy templates, hidden behind a new API microversion. The endpoint will support standard CRUD operations.

In the following API, a UUID or trait name is accepted for a deploy templates identity.

List all

List all deploy templates:

```
GET /v1/deploy-templates
```

Request: empty

Response:

```
{
  "deploy-templates": [
    {
      "name": "CUSTOM_BM_CONFIG_RAID_DISK_MIRROR",
      "steps": [
        {
          "interface": "raid",
          "step": "create_configuration",
          "args": {
            "logical_disks": [
              {
                "size_gb": "MAX",
                "raid_level": "1",
                "is_root_volume": true
              }
            ],
            "delete_configuration": true
          },
          "priority": 10
        }
      ],
      "uuid": "8221f906-208b-44a5-b575-f8e8a59c4a84"
    },
    {
      ...
    }
  ]
}
```

Response codes: 200, 400

Policy: admin or observer.

Show one

Show a single deploy template:

```
GET /v1/deploy-templates/<deploy template ident>
```

Request: empty

Response:

```
{
  "name": "CUSTOM_BM_CONFIG_RAID_DISK_MIRROR",
  "steps": [
    {
      "interface": "raid",
      "step": "create_configuration",
      "args": {
        "logical_disks": [
          {
```

(continues on next page)

(continued from previous page)

```
            "size_gb": "MAX",
            "raid_level": "1",
            "is_root_volume": true
          }
        ],
        "delete_configuration": true
      },
      "priority": 10
    }
  ],
  "uuid": "8221f906-208b-44a5-b575-f8e8a59c4a84"
}
```

Response codes: 200, 400, 404

Policy: admin or observer.

Create

Create a deploy template:

```
POST /v1/deploy-templates
```

Request:

```
{
  "name": "CUSTOM_BM_CONFIG_RAID_DISK_MIRROR",
  "steps": [
    {
      "interface": "raid",
      "step": "create_configuration",
      "args": {
        "logical_disks": [
          {
            "size_gb": "MAX",
            "raid_level": "1",
            "is_root_volume": true
          }
        ],
        "delete_configuration": true
      },
      "priority": 10
    }
  ],
}
```

Response: as for show one.

Response codes: 201, 400, 409

Policy: admin.

Update

Update a deploy template:

```
PATCH /v1/deploy-templates/{deploy template ident}
```

Request:

```
[
  {
    "op": "replace",
    "path": "/name"
    "value": "CUSTOM_BM_CONFIG_RAID_DISK_MIRROR"
  },
  {
    "op": "replace",
    "path": "/steps"
    "value": [
      {
        "interface": "raid",
        "step": "create_configuration",
        "args": {
          "logical_disks": [
            {
              "size_gb": "MAX",
              "raid_level": "1",
              "is_root_volume": true
            }
          ],
          "delete_configuration": true
        },
        "priority": 10
      }
    ]
  }
]
```

Response: as for show one.

Response codes: 200, 400, 404, 409

Policy: admin.

The `name` and `steps` fields can be updated. The `uuid` field cannot.

Delete

Delete a deploy template:

```
DELETE /v1/deploy-templates/{deploy template ident}
```

Request: empty

Response: empty

Response codes: 204, 400, 404

Policy: admin.

Client (CLI) impact

ironic CLI

None

openstack baremetal CLI

In each of the following commands, a UUID or trait name is accepted for the deploy templates identity.

For the `--steps` argument, either a path to a file containing the JSON data or `-` is required. If `-` is passed, the JSON data will be read from standard input.

List deploy templates:

```
openstack baremetal deploy template list
```

Show a single deploy template:

```
openstack baremetal deploy template show <deploy template ident>
```

Create a deploy template:

```
openstack baremetal deploy template create --name <trait> --steps <deploy␣
↪steps>
```

Update a deploy template:

```
openstack baremetal deploy template set <deploy template ident> [--name
↪<trait] [--steps <deploy steps>]
```

Delete a deploy template:

```
openstack baremetal deploy template delete <deploy template ident>
```

In these commands, `<deploy steps>` are in JSON format and support the same input methods as clean steps - string, file or standard input.

RPC API impact

None

Driver API impact

None

Nova driver impact

Existing traits integration is enough, only now the selected traits on boot become more important.

Ramdisk impact

None

Security impact

Allowing the deployment process to be customised via deploy templates could open up security holes. These risks are mitigated, as seen through the following observations:

- Only admins can define the set of allowed traits for each node.
- Only admins can define the set of requested traits for each Nova flavor, and allow access to that flavor for other users.
- Only admins can create or update deploy templates via the API.
- Deploy steps referenced in deploy templates are defined in driver code.

Other end user impact

Users will need to be able to discover what each Nova flavor does in terms of deployment customisation. Beyond checking requested traits and cross-referencing with the ironic deploy templates API, this is deemed to be out of scope. Operators should provide sufficient documentation about the properties of each flavor. The ability to look up a deploy template by trait name should help here.

Scalability impact

Increased activity during deployment could have a negative impact on the scalability of ironic.

Performance Impact

Increased activity during deployment could have a negative impact on the performance of ironic, including increasing the time required to provision a node.

Other deployer impact

Deployers will need to ensure that Nova flavors have required traits set appropriately.

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
Mark Goddard (mgoddard)

Other contributors:

- Dmitry Tantsur (dtantsur)
- Ruby Loo (rloo)

Work Items

- Add DB tables and objects for deploy templates
- Write code to map traits to deploy templates
- Extend node validation to check all deploy templates are valid
- Add API to add deploy templates
- Extend CLI to support above API
- Write tests

Dependencies

- Node traits spec http://specs.openstack.org/openstack/ironic-specs/specs/approved/node-traits.html
- Deploy steps spec http://specs.openstack.org/openstack/ironic-specs/specs/11.1/deployment-steps-framework.html

Testing

Unit tests will be added to ironic. Tempest API tests will exercise the deploy templates CRUD API.

Upgrades and Backwards Compatibility

The deploy steps API endpoint will be hidden behind a new API version.

During normal operation when the ironic conductor is not pinned, deploy templates will be used to add deploy steps during node provisioning, even if the caller of the node state API uses a microversion that does not support deploy templates.

During an upgrade when the ironic conductor is pinned, deploy templates will not be used to add deploy steps during node provisioning.

Documentation Impact

- Admin guide on how to configure Nova flavors and deploy templates
- Update API ref
- Update CLI docs

References

- http://specs.openstack.org/openstack/ironic-specs/specs/11.1/deployment-steps-framework.html
- http://specs.openstack.org/openstack/ironic-specs/specs/approved/node-traits.html

Huawei iBMC Driver

https://storyboard.openstack.org/#!/story/2004635

This specification proposes to add new interfaces that provide Ironic support to Huawei iBMC 2288H V5, CH121 V5 series servers.

Problem description

Huaweis Intelligent Baseboard Management System (iBMC) is an embedded server management system that is used to manage servers throughout their lifecycle. It provides a series of management tools for hardware status monitoring, deployment, energy savings, and security protection.

In addition to managing the nodes using IPMI protocol, this specification proposes to add hardware types and interfaces to manage Huawei servers using iBMC REST API.

Proposed change

New hardware type named *ibmc* will be added as part of this change. New power, management and vendor interfaces will be implemented for the *ibmc* hardware.

The interfaces use iBMC REST API to communicate with iBMC. The interfaces used are:

- iBMC.IBMCPower for Power operations
- iBMC.IBMCManagement for Management operations
- iBMC.IBMCVendor for Vendorspecific operations
- Power:

  This feature allows the user to turn the node on/off or reboot by using the power interface which will in turn call iBMC REST API.

- Management:

  This feature allows the user to get and set the primary boot device of the Huawei servers, and to get the supported boot devices.

- Vendor:

  This feature allows the user to perform vendor specific operations. For example, query the boot up sequence of the Huawei servers.

```
$ openstack baremetal node passthru call --http-method GET \
  <node id or node name> boot_up_seq
$ ["Pxe", "Hdd", "Cd", "Others"]
```

Alternatives

None

Data model impact

None

RPC API impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

Driver API impact

None

Nova driver impact

None

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Ramdisk impact

None

Other deployer impact

The following driver_info fields are required while enrolling node into Ironic:

- ibmc_address: The URL address to the ibmc controller, example: https://example.com
- ibmc_username: User account with admin/server-profile access privilege
- ibmc_password: User account password

- ibmc_verify_ca(optional): Whether to verify the host certificate or the path of a certificate file or directory with trusted certificates

Developer impact

None

Implementation

Assignee(s)

Primary assignee:

- QianBiao Ng (iampurse@vip.qq.com)
- Bill Chan (biaocy91@gmail.com)

Other contributors:
None

Work Items

- Add new iBMC hardware type, and adding new interfaces for Power, Management and Vendor.
- Writing appropriate unit tests to provide test coverage for iBMC driver.
- Writing configuration documents.
- Building a third party CI.

Dependencies

- Use python-ibmcclient library (not released) to communicate with HUAWEI iBMC REST API.

Testing

- Unit tests will be implemented for new iBMC driver.
- Third party CI will be provided.

Upgrades and Backwards Compatibility

None

Documentation Impact

- Updating Ironic documentation section *Enabling Drivers* with iBMC related instructions.

References

None

Out-of-band RAID configuration for Gen10 and above HPE Proliant Servers

https://bugs.launchpad.net/ironic/+bug/1716329

This specification proposes implementation of out-of-band RAID configuration for ILO managed HPE Proliant servers.

Problem description

In the current scenario where RAID configuration on HPE Proliant servers is done only via inband cleaning, Ilo5 based HPE Proliant Gen10 servers provide support to perform out-of-band RAID configuration which was not there in Gen9 and below servers. However, the raid creation or deletion will take into effect only when the system reaches POST stage. Hence, creation or deletion of RAID needs to be accompanied by a reboot.

Proposed change

This spec proposes to implement out-of-band RAID configuration as described by the parent spec [1]. This will require the implementation of a new hardware type `Ilo5Hardware` and a new raid interface for ilo as `IloRAID`.

OOB RAID configuration will be a four step process. 1. delete_configuration - delete the current raid config from the system. 2. read_configuration - get the updated raid config from system and update the node properties accordingly. 3. create_configuration - create the raid config which set by the user in target_raid_config of node properties. 4. read_configuration - get the updated raid config from system and update the node properties accordingly.

List of changes required:

- The following would be the composition of `Ilo5Hardware`:
  - This hardware type would be supported on ilo5 based HPE Proliant servers.
  - Ilo5Hardware will inherit all interfaces of parent class IloHardware.
  - Ilo5Hardware will support the new RAID interface `IloRAID`.
- The following would be the composition of `IloRAID`:
  - IloRAID will inherit RAIDInterface of base class.
  - `delete_configuration` - This will delete the RAID configuration on the bare metal node.
    - Since a reboot is required for changes to get reflected, this function will be decorated with additional argument *reboot_required* with value set to *True*.
    - It will create an IloClient object from proliantutils library to do operations on the iLO. This will make call to delete_raid_configuration of proliantutils library to delete the logical drives on the system.
  - `create_configuration` - This will create the RAID configuration on the bare metal node.
    - Since a reboot is required for changes to get reflected, this function will be decorated with additional argument *reboot_required* with value set to *True*.
    - It will create an IloClient object from proliantutils library to do operations on the iLO. This will make call to create_raid_configuration of proliantutils library to place a request to firmware to create the logical drives on the system.
  - `read_configuration` - This will read the RAID configuration on the bare metal node.

        * It will create an IloClient object from proliantutils library to do operations on the iLO. This will make call to read_raid_configuration of proliantutils library to get the logical drives on the system. Hence, it will update the node properties with the actual RAID configuration when called after `create_configuration` and to `None` when called after `delete_configuration`.

- The following would be the updates required in cleaning architecture in ironic to support post reboot operation if required any clean step.
  - Addition a new boolean positional argument `reboot_required` to clean_step function of BaseInterface. Default is set to `False` for this parameter. NOTE: The same approach is being used in inband cleaning for steps that require reboot.
  - Update ironic/conductor/manager.py:_do_next_clean_step() for each step to call prepare_cleaning() if reboot_required is set to True and result of the last command interface.execute_clean-step() is not `clean wait`.

Alternatives

One can perform in-band raid configuration to achieve the same result. However, The ramdisk to be used in such case should have proliant-tools element that bundles ssacli utility required for RAID operations as part of the image.

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

None

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

User need to configure below two things to make use of OOB RAID configuration on HPE Proliant Gen10 servers.

- Configure the new hardware type `ilo5` to ([DEFAULT] `enabled_hardware_types`).
- Configure the new raid interface `ilo5` to ([DEFAULT] `enabled_raid_interfaces`).

Developer impact

None

Implementation

Assignee(s)

Primary assignee: theanshuljain

Work Items

- Add a new hardware type for ilo `Ilo5Hardware` which inherits IloHardware.
- Add a new hardware interface `IloRAID` which inherits base.RAIDInterface.
- Writing unit-test cases for the new OOB RAID interface.

Dependencies

Support for OOB RAID in proliantutils is under development and is yet to be released.

Testing

Unit test cases will be added. Will be tested in 3rd party CI setup.

Upgrades and Backwards Compatibility

None

Documentation Impact

Need to update iLO driver documentation for new hardware type and RAID interface.

References

[1] Ironic generic raid spec: https://review.opendev.org/173214

Ownership Information Storage

https://storyboard.openstack.org/#!/story/2001814

In many large businesses with hardware fleets, there may be a variety of ownership of the underlying hardware for tracking purposes. A good example of this is in a hybrid service provider scenario where an operator may directly own a portion of the hardware, may have the hardware on lease, and may ultimately have customer owned hardware.

We simply cannot model this with existing node fields, since there may be resource sharing agreements also in place between the owners of the hardware. And as such, we need some way to store and represent the end owner of the hardware for tracking and logistical purposes.

Problem description

While scenarios differ, ultimately there is a need to be able to store information in a top level fashion about who owns a given piece of hardware.

Information of this sort can be vital when it comes time for tax asset inventories, or just simple tracking of where the hardware came from.

Providing the ability to search and return the hardware that is known to be owned by a particular group allows for faster correlation of the disposition of the hardware for auditing and accounting purposes.

Proposed change

Proposing to add a new informational field to the node object that can be queried via the REST API, and stored in the database. No other initial use of this field is expected.

Future use could also be automatic in the scenario if there is a driver that is aggregating a number of management systems, however that is out of scope.

Alternatives

An operator could potentially store this information in extra, however then they would still need to dump all of the nodes out to obtain a list of nodes with the specific information that is needed. The operator would begin to hit limits with the number of responses from the API, and would need to likely create their own tooling around list processing.

Data model impact

A `owner` field would be added to the nodes table as a `VARCHAR(255)` and will have a default value of `null` in the database.

State Machine Impact

None

REST API impact

An `owner` field will be returned as part of the node. The API shall be updated to allow a user to set and unset the value via the API.

Additionally the GET syntax for the nodes list will be updated to allow a list of matching nodes to be returned.

POST/PATCH operations to the field will be guarded by a new microversion. The field shall not be returned unless the sufficient microversion is supplied for GET operations.

Client (CLI) impact

ironic CLI

None

openstack baremetal CLI

A corresponding change will be necessary to enable the ability for a user to set/unset the value from a command line.

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

None

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

None

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
TheJulia - juliaashleykreger@gmail.com

Other contributors:
None

Work Items

- Add database field.
- Add object field.
- Add REST API functionality and microversion.
- Update REST API documentation.
- Update python-ironicclient.

Dependencies

None

Testing

Basic API CRUD testing will be added. There is no need for additional testing as this is an informational field for the API user/baremetal operator.

Upgrades and Backwards Compatibility

Field will be created as part of the upgrade process with a default value in the database schema.

Documentation Impact

REST API documentation will need to be updated.

References

None

Smart NIC Networking

https://storyboard.openstack.org/#!/story/2003346

This spec describes proposed changes to Ironic to enable a generic, vendor-agnostic, baremetal networking service running on smart NICs, enabling baremetal networking with feature parity to the virtualization use-case.

Problem description

While Ironic today supports Neutron provisioned network connectivity for baremetal servers through an ML2 mechanism driver, the existing support is based largely on configuration of TORs through vendor-specific mechanism drivers, with limited capabilities.

Proposed change

There is a wide range of smart/intelligent NICs emerging on the market. These NICs generally incorporate one or more general purpose CPU cores along with data-plane packet processing acceleration, and can efficiently run virtual switches such as OVS, while maintaining the existing interfaces to the SDN layer.

The proposal is to extend Ironic to enable use of smart NICs to implement generic networking services for Bare Metal servers. The goal is to enable running the standard Neutron Open vSwitch L2 agent, providing a generic, vendor-agnostic bare metal networking service with feature parity compared to the virtualization use-case. The Neutron Open vSwitch L2 agent manages the OVS bridges on the smart NIC.

In this proposal, we address two use-cases:

1. Neutron OVS L2 agent runs locally on the smart NIC.

   This use case requires a smart NIC capable or running openstack control services such as the Neutron OVS L2 agent. This use case strives to view the smart NIC as an isolated hypervisor for the baremetal node, with the smart NIC providing the services to the bare metal image running on the host (as a hypervisor would provide services to a VM). While this spec initially targets Neutron OVS L2 agent, the same implementation would naturally and easily be extended to any other ML2 plugin as well as to additional agents/services (for example exposing emulated NVMe storage devices back-ended by a storage initiator on the smart NIC).

2. Neutron OVS L2 agent(s) run remotely and manages the OVS bridges for all the baremetal smart NICs.

The enhancements for Neutron OVS L2 agent captured in[1],[2] and[3].

- Set the smart NIC configuration

  smart NIC configuration includes the following:

  1. extend the ironic port with is_smartnic field. (default to False)
  2. smart NIC hostname - the hostname of server/smart NIC where the Neutron OVS agent is running. (required)
  3. smart NIC port id - the port name that needs to be plugged to the integration bridge. B in the diagram below (required)
  4. smart NIC SSH public key - ssh public key of the smart NIC (required only for remote)
  5. smart NIC OVSDB SSL certificate - OVSDB SSL of the OVS in smart NIC (required only for remote)

  The OVS ML2 mechanism driver will determine if the Neutron OVS Agent runs locally or remotely based on smart NIC configuration passed from ironic. The config attribute will be stored in the local_link_information of the baremetal port.

  In the scope of this spec the smart NIC config will be set manually by the admin.

- Deployment Interfaces

  Extending the ramdisk, direct, iscsi and ansible to support the smart nic use-cases.

  The Deployment Interfaces call network interface methods such as: add_provisioning_network, remove_provisioning_network, configure_tenant_networks, unconfigure_tenant_networks, add_cleaning_network and remove_cleaning_network.

  These network methods are currently ordinarily called when the baremetal is powered down, ensuring proper network configuration on the TOR before booting the bare metal.

  smart NICs share the power state with the baremetal, requiring the baremetal to be powered up before configuring the network. This leads to a potential race where the baremetal boots and access the network prior to the network being properly configured on the OVS within the smart NIC.

  To ensure proper network configuration prior to baremetal boot, the deployment interfaces will intermittently boot the baremetal into the BIOS shell, providing a state where the ovs on the smart NIC may be configured properly before rebooting the bare metal into the actual guest image or ramdisk. The ovs on the smart NIC will get programmed after we verify that the neutron ovs agent is alive.

  The following code for configure/unconfigure network:

```
if task.driver.network.need_power_on(task):
    old_power_state = task.driver.power.get_power_state(task)
    if old_power_state == states.POWER_OFF:
        # set next boot to BIOS to halt the baremetal boot
        manager_utils.node_set_boot_device(task, boot_devices.BIOS,
                                           persistent=False)
        manager_utils.node_power_action(task, states.POWER_ON)
```

(continues on next page)

[1] https://review.opendev.org/#/c/619920/
[2] https://review.opendev.org/#/c/595402/
[3] https://review.opendev.org/#/c/595512/

(continued from previous page)

```
# ...
# call task.driver.network method(s)
# ...

if task.driver.network.need_power_on(task):
    manager_utils.node_power_action(task, old_power_state)
```

The following methods in the deployment interface are calling to one or more configure/unconfigure networks and should be updated with the logic above.

- iscsi Deploy Interface
  - iscsi_deploy::prepare
  - iscsi_deploy::deploy
  - iscsi_deploy::tear_down
- ansible Deploy Interface
  - ansible/deploy::reboot_and_finish_deploy
  - ansible/deploy::prepare
  - ansible/deploy::tear_down
  - ansible/deploy::prepare_cleaning
  - ansible/deploy::tear_down_cleaning
- direct Interface
  - agent::prepare
  - agent::tear_down
  - agent::deploy
  - agent::rescue
  - agent::unrescue
  - agent_base_vendor::reboot_and_finish_deploy
  - agent_base_vendor::_finalize_rescue
- RAM Disk Interface
  - pxe::deploy
- Common cleaning methods
  - deploy_utils::prepare_inband_cleaning
  - deploy_utils::tear_down_inband_clean

- Network Interface

  Extend the base *network_interface* with need_power_on - return true if any ironic port attached to the node is a smart nic

  Extend the ironic.common.neutron add_ports_to_network/ remove_ports_from_network methods for the smart NIC case:

  - on add_ports_to_network and has smartNIC do the following:
    * check neutron agent alive - verify that neutron agent is alive
    * create neutron port
    * check neutron port active - verify that neutron port is in active state
  - on remove_ports_from_network and has smartNIC do the following:
    * check neutron agent alive - verify that neutron agent is alive
    * delete neutron port
    * check neutron port is removed

- Neutron ml2 OVS changes:
  - Introduce a new vnic_type for `smart-nic`.
  - Update the Neutron ml2 OVS to bind smart-nic vnic_type with *binding:profile* smart NIC config.
- Neutron OVS agent changes:

Example of smart NIC model:

```
+---------------------+
|      baremetal      |
| +-----------------+ |
| |  OS Server    | | |
| |               | | |
| |      +A       | | |
| +------|--------+ | |
|        |          | |
| +------|--------+ | |
| |  OS SmartNIC  | | |
| |    +-+B-+     | | |
| |    |OVS |     | | |
| |    +-+C-+     | | |
| +------|--------+ | |
+--------|----------+
         |

A - port on the baremetal host.
B - port that represents the baremetal port in the smart NIC.
C - port that represents to the physical port in the smart NIC.

Add/Remove Port B to the OVS br-int with external-ids

In our case we will use the neutron OVS agent to plug the port on update
port event with the following external-ids: iface-id,iface-status, attached-
↪mac
and node-uuid
```

Alternatives

- Delay the Neutron port binding (port binding means setting all the OVSDB/Openflows config on the SmartNIC) to be performed by Neutron later (once the bare metal is powered up). The problem with this approach is that we have no guarantee of if/when the rules will be programmed, and thus may inadvertently boot the baremetal while the smart NIC is still programmed on the old network.

Data model impact

A new `is_smartnic` boolean field will be added to Port object.

State Machine Impact

None

REST API impact

The port REST API will be modified to support the new `is_smartnic` field. The field will be readable by users with the baremetal observer role and writable by users with the baremetal admin role.

Updates to the is_smartnic field of ports will be restricted in the same way as for other connectivity related fields (link local connection, etc.) - they will be restricted to nodes in the `enroll`, `inspecting` and `manageable` states.

Client (CLI) impact

ironic CLI

None

openstack baremetal CLI

The openstack baremetal CLI will be updated to support getting and setting the `is_smartnic` field on ports.

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

None

Security impact

- Smart NIC Isolation

Both use cases run infrastructure functionality on the smart NIC, with the first use case also running control plane functionality.

This requires proper isolation between the untrusted bare metal host and the smart NIC, preventing any/all direct or indirect access, both through the network interface exposed to the host and through side channels such as the platform BMC.

Such isolation is implemented by the smart NIC device and/or the hardware platform vendor. There are multiple approaches for such isolation, ranging from completely physical disconnection of the smart NIC from the platform BMC to a platform with a trusted BMC wherein the BMC considers the baremetal host an untrusted entity and restricts its capabilities/access to the platform.

In the absence of such isolation, the untrusted baremetal tenant may be able to gain access to the provisioning network, and in the second may be able to compromise the control plane.

Proper isolation is dependent on the platform hardware/firmware, and cannot be directly enforced/guaranteed by ironic. Users of smart NIC use case should be made well aware of this via explicit documentation, and should be guided to verify the proper isolation exists on their platform when enabling such use cases.

- Security Groups

This will allow to use Neutron OVS agent pipeline. One of the features in the pipeline is security groups which will enhance the security model when using baremetal in a cloud.

- Security credentials

The node running the Neutron OVS agent (smart NIC or remote, according to use case) should be configured with the message bus credentials for the Neutron server.

In addition, for the second use case, the SSH public key and OVSDB SSL certificate should be configured for the smart NIC port.

Other end user impact

- Baremetal admin needs to update the SmartNIC config manually.

Scalability impact

None

Performance Impact

None

Other deployer impact

None

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
hamdyk - hamdy@mellanox.com

Work Items

- Update the Neutron network interface to populate the Smart NIC config from the ironic port to the Neutron port *binding:profile* attribute.
- Update the network_interface and common.neutron as described above
- Update deployment interfaces as described above
- Documentation updates.

Dependencies

None, but the Neutron specs[Page 334, 1],[Page 334, 2] and[Page 334, 3] depend on this spec.

Testing

- Mellanox CI Jobs testing with Bluefield SmartNIC

Upgrades and Backwards Compatibility

None

Documentation Impact

- Update the multitenancy.rst with setting the SmartNIC config
- Document the security implications/guidelines under admin/security.rst

References

5.6.2 12.0

The direct deploy interface provisioning with HTTP server

https://storyboard.openstack.org/#!/story/1598852

This spec proposes a mechanism to provision baremetal nodes by hosting custom HTTP service as an image source provider to the `direct` deploy interface, when the Image service is utilized.

Problem description

Currently the `direct` deploy interface requires an unauthenticated image source link, so that the agent running at ramdisk can download the image from provided link and deploy it to the hard disk.

In a typical deployment, user images are managed by the Image service and in most cases, access is controlled by the Identity service. The `direct` deploy interface relies on the Object Storage service to generate an unauthenticated URL which is accessible for a period (i.e. tempurl).

The problem is the Object Storage service is not always adopted in a deployment due to various reasons, and itself imposes restrictions on deployment. E.g.:

- It has little benefit for a small cloud but takes more hardware resource.
- It requires baremetal nodes to have access to control plane network, which is a restriction to network topology.
- It requires the Image service be configured with a backend of swift, which may conflicts with original one.

As there is no mechanism or means for ironic to leverage a local HTTP server to provide temporary image file for IPA to facilitate a node deployment, this proposal is to offer an alternative by providing such support.

Proposed change

An HTTP server on the ironic conductor node is required for this feature to work.

Currently there are two scenarios, if the `instance_info['image_source']` indicates its a glance image, the `direct` deploy interface generates tempurl via glance client, and stores it to `instance_info['image_url']`, otherwise it will be directly taken as `image_url`. The two cases typically represent using the Bare Metal service in the cloud or as a standalone service, respectively.

The proposal introduces a new string option `[agent]image_download_source` to control which kind of image URL will be generated when the `image_source` is a glance image. Allowed values are `swift` and `http`, defaults to `swift`.

The process of the `direct` deploy interface on different configurations is defined as:

- `swift`: Keeps current logic, generates tempurl and update it to `instance_info['image_url']`.
- `http`: Downloads instance image via `InstanceImageCache` before node deployment, creates symbolic link to downloaded instance image in the directory accessible by local HTTP service, generates proper URL and updates it to `instance_info['image_url']`.

The existing `[deploy]http_root` is reused for storing symbolic links to downloaded instance images. A new string option `[deploy]http_image_subdir` is introduced to keep it isolated with iPXE related scripts. The default value is `agent_images`. The existing `[deploy]http_url` is reused to generate instance image URLs.

The `direct` deploy interface will use the same instance cache for image caching, the caching will be performed at `AgentDeploy.deploy`. After an instance image is cached, the `direct` deploy interface creates a symbolic link at `<http_root>/<http_image_subdir>` to reference the instance image. It will be `/httpboot/agent_images/<node-uuid>` if all goes to default.

The `direct` deploy interface generates URL for the instance image and updates it to `instance_info` at `AgentDeploy.prepare`. The corresponding image URL will be `<http_url>/<http_image_subdir>/<node-uuid>`. The symbolic link will be removed at `AgentDeploy.deploy`

when a node deploy is done, or `AgentDeploy.clean_up` when a node is teared down from the state `deploy failed`.

Rule to convert image

Currently the `iscsi` deploy interface will convert image to `raw` if `[DEFAULT]force_raw_images` is set to True.

While IPA treats instance image in two different ways:

- If the instance image format is `raw`, `stream_raw_images` is True and image type is whole disk image, the image will be streamed into the target disk of the Bare Metal.
- Otherwise the image will be cached into memory before written to disk.

To avoid a raw image been cached into the memory of Bare Metal, the `direct` deploy interface will convert image to raw only if following criteria is met:

- `[DEFAULT]force_raw_images` is set to True,
- `[agent]stream_raw_images` is set to True,
- The instance image type is a whole disk image.

The `direct` deploy interface will recalculate MD5 checksum and update necessary fields to `instance_info` if image conversion happened.

Cache sharing

`iscsi` and `direct` deploy interface are sharing the same cache, but apply different rule to whether the image should be converted to raw. It leads to cache compatibility issue when both interface are in use.

As an example, suppose we deploy node A (using iscsi) with a partition image, then deploy node B (use direct) with the same image. The image in the cache is converted to raw, but according to the rule of `direct` deploy interface, it assumes image will not be converted to raw, though it specifies `force_raw` to false to the image cache, due to cache hit, actually no image action will be performed, this will leads to the situation that the `direct` deploy interface actually provide a raw image but without MD5 recalculation.

Vice versa, if we reverse the order above, the `iscsi` deploy interface may get a qcow with `[DEFAULT]force_raw_images` set to true, though its probably not an issue because populate_image will check image format before writing. its still not a consistent behavior.

To address the issue described above, this spec proposes to update `ImageCache.fetch_image` to take the input argument `force_raw` into account for the master image file name:

- The master file name is not changed if `force_raw` is set to `False`.
- The master file name will have `.converted` as file extension if `force_raw` is set to `True`, e.g.:

```
/var/lib/ironic/master_images/6e2c5132-db24-4e0d-b612-478c3539da1e.
↪converted
```

Note that the `.converted` extension merely acts as an indicator that the image downloaded has gone through the conversion logic. For a raw image in the glance, the name of master image file still has `.converted` as long as `force_raw` argument passed in is True.

Alternatives

Another implementation approach Support all glance backends in the agent is to support IPA directly downloading instance image from glance, the deployment restriction of this approach is the same as agent today, baremetal has to access glance at provisioning network. But it can address the dependency issue on glance backend, thus can be a further work.

Instead of supporting HTTP provisioning from the `direct` deploy interface, it can also be implemented as a new deploy interface, `direct-http` for example.

Data model impact

None

State Machine Impact

None.

REST API impact

None

Client (CLI) impact

None.

ironic CLI

openstack baremetal CLI

RPC API impact

None.

Driver API impact

None.

Nova driver impact

None

Ramdisk impact

None

Security impact

Providing HTTP service on ironic conductor node will expose accessible port thus can be a security impact. There are several ways to improve security:

1. Bind the port to the network interface dedicated to provisioning networks.
2. Configure firewall to prevent access from source IP addresses other than the provisioning networks.

There might be other ways, but thats beyond the scope of this spec.

To allow HTTP server accessing instance image in the cache directory, the file-creation mask of user for ironic conductor service should be configured to be accessible by the user of HTTP service. Most systems use 022 or 002 as the default umask, it should be sufficienth. There would be a security impact if its not the case.

Other end user impact

None

Scalability impact

Instance images will be cached on the ironic conductor node once the `[agent]image_download_source` is set to `http`, it will cost more disk space if the conductor node is using `direct` deploy interface before. The expected space usage basically should be no more than `iscsi` deploy interface.

IPA downloads instance image directly from the conductor node, which will reduce traffic on the control plane network, by the cost of increasing traffic on each conductor node. The consumption should be no more than `iscsi` deloy interface.

Performance Impact

Depending on the hardware and image type, recalculating MD5 checksum for a raw image could consume considerable amount of CPU/IO resources. If the performance on ironic conductor node is in concern, please set `[DEFAULT]force_raw_images` to `False` (The option is `True` by default).

Other deployer impact

When using this feature, an HTTP server should be set up and configured on each ironic conductor node.

Each HTTP servers should be configured to follow symlinks for instance images are accessible from external requests. Refer to `FollowSymLinks` if Apache HTTP server is used, or `disable_symlinks` if Nginx HTTP server is used.

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
kaifeng

Other contributors:
sambetts

Work Items

- Promote instance cache to be a global cache, usable for other interfaces.
- Implement the proposed work for `direct` deploy interface, includes image caching, checksum recalculating, symlink management, etc.
- Update documents.

Dependencies

None

Testing

This feature will be covered by unit test.

Upgrades and Backwards Compatibility

Two new options `[agent]image_download_source` and `[deploy]http_image_subdir` are introduced in this feature.

`[agent]image_download_source` defaults to `swift`, which should have no impact on upgrades.

The change of the cache file naming could probably invalidate some cached instance images after upgrades, they will be re-cached when used, images not referenced will be cleaned up eventually. This will have no impact if caching is disabled before upgrade.

Documentation Impact

Update admin/interfaces/deploy.rst to describe the usage of this feature.

References

None

Expose conductor information from API

https://storyboard.openstack.org/#!/story/1724474

The spec proposes to expose conductor information from API, as well as the mapping between conductors and nodes.

Problem description

In deployment with multiple ironic conductors incorporated, nodes are distributed across conductors by hashring, currently we have no mechanism to identify the specific conductor for a given node without database query.

There are several cases we need to locate the host of the conductor for a node, for example:

- A node failed to boot from PXE during node deploy, we need to check the PXE environment of the conductor host.
- Ramdisk logs are stored in the conductor host if the backend is set to local, this requires identifying the exact conductor which serves a node.

Meanwhile we are lacking support from API for information related with conductors, which might be useful for diagnostic. For example:

- How many conductors in this cloud, whether they are alive.
- The conductor group each conductor belongs to.
- Nodes currently mapped to this conductor.

This information also makes it possible to implement features like service monitoring, alarming from the outside of ironic.

Proposed change

Adds an API endpoint `/v1/conductors` to retrieve a list of conductor information, including the hostname, conductor group, and alive flag.

The alive flag is calculated in the same way as how ironic assumes a conductor is not alive, by checking `update_at` and `heartbeat_timeout`.

Adds API endpoint `/v/conductors/{hostname}` to retrieve detailed conductor information, including nodes affiliated with this conductor. The mapping is retrieved from hashring.

Adds a field `conductor` to the Node API object, which indicates the hostname of the conductor currently the node is mapped to.

Alternatives

Admins can directly query the ironic database for a conductor list. To locate the last conductor that served a node, query nodes table for the `conductor_affinity` field, then query `conductors` table for the hostname of the conductor. The `conductor_affinity` is updated in several cases, mainly during node deployment time. There is no alternative to get the real time mapping from the hashring.

Data model impact

None

State Machine Impact

None

REST API impact

API microversion will be bumped.

Add a new `conductor` API object, and three API endpoints:

- GET /v1/conductors
  - Get the list of conductors known by ironic
  - Client with earlier microversion before this feature implemented will receive 404. For a normal request, 200 is returned.
  - Sample response data:

```
{
  "conductors": [
    {
      "hostname": "Compute01",
      "conductor_group": "region1",
      "alive": true,
      "links": [
        {
          "href": "http://127.0.0.1:6385/v1/conductors/Compute01",
          "rel": "self"
        },
        {
          "href": "http://127.0.0.1:6385/conductors/Compute01",
          "rel": "bookmark"
        }
      ]
    },
    {
      "hostname": "Compute03",
      "conductor_group": "region1",
      "alive": true,
      "links": [...]
    }
  ]
}
```

- GET /v1/conductors/{hostname}
  - Get detailed information of a conductor by hostname
  - Client with earlier microversion before this feature implemented will receive 404. For a normal request, 200 is returned.
  - Sample response data:

```
{
  "hostname": "Compute01",
  "conductor_group": "region1",
  "alive": true
  "drivers": ["ipmi"],
  "last_seen": "2018-09-10 19:53:17"
  "links": [
    {
      "href": "http://127.0.0.1:6385/v1/conductors/Compute01",
      "rel": "self"
    },
    {
      "href": "http://127.0.0.1:6385/conductors/Compute01",
      "rel": "bookmark"
    }
  ]
}
```

- GET /v1/nodes/{node_ident}/conductor
  - Get detailed information of a conductor for given node
  - Client with earlier microversion before this feature implemented will receive 404. For a normal request, 200 is returned.
  - The response data is same as /v1/conductors/{hostname}

Change `Node` API object in the following way:

- Add a read-only attribute `conductor` to indicate the hostname of associated conductor.
- Retrieve and assign the conductor hostname in `Node.convert_with_links`.

The hostname of the conductor will be returned in these endpoints:

- `POST /v1/nodes`
- `GET /v1/nodes` (when `detail` is set to true)
- `GET /v1/nodes/detail`
- `GET /v1/nodes/{node_ident}`

Sample response data for `GET /v1/nodes/{node_ident}` would be:

```
{
  "nodes": [
    {
      "instance_uuid": null,
      "conductor": "Compute01",
      "uuid": "a308bca6-e6a3-4349-b8ea-695e17672898",
      "links": ...,
      "maintenance": false,
      "provision_state": "available",
      "power_state": "power off",
      "name": "node-0"
    }
  ]
}
```

Add support for querying nodes by conductor hostname for endpoints:

- `GET /v1/nodes`
- `GET /v1/nodes/detail`

For example, `GET /v1/nodes/?conductor=Compute01` will return nodes mapped to the conductor whose hostname is Compute01.

Client (CLI) impact

ironic CLI

None

openstack baremetal CLI

Enhance ironic client with two new command:

- `openstack baremetal conductor list`
- `openstack baremetal conductor show <hostname>`

Expose the conductor field in command:

- `openstack baremetal node list --detail`
- `openstack baremetal node show <node>`

Support node querying by conductor hostname:

- `openstack baremetal node list --conductor <hostname>`

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

None

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

None

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
kaifeng <kaifeng.w@gmail.com>

Work Items

- Implement the API endpoints.
- Add `conductor` field to node API object.
- CLI enhancement for API changes.
- Documentation and api reference.

Dependencies

None

Testing

The feature will be covered by unit tests.

Upgrades and Backwards Compatibility

The feature will be guarded by microversion.

Documentation Impact

New APIs will be documented in the API reference. New commands will be documented in the ironic-client documentation.

References

None

Promote iPXE to separate boot interface

https://bugs.launchpad.net/ironic/+bug/1628069

The iPXE boot should be promoted to a separate boot driver interface. This will simplify the code base and allow to not force global PXE *vs* iPXE conductor setting.

Problem description

Currently setting PXE or iPXE is enforced per-conductor, and taking into account the possibility of node take-over, this choice is effectively global for the whole Ironic installation.

Due to this the current code of `PXEBoot` interface is riddled with constructs of:

```
if CONF.pxe.ixpe_enabled ...
```

Recently added or proposed changes (like `CONF.pxe.ipxe_use_swift` option) make the logic there even more complicated.

Proposed change

It is proposed to implement a separate iPXE boot interface, which will use the new way of serving iPXE boot scripts and boot config files directly from Ironic API as outlined in dynamic iPXE configuration spec.

The new interface will get a separate `[ipxe]` config section, where all `[pxe]ipxe_*` options should be moved following proper deprecation policy for config options. Current iPXE-related behavior of `PXEBoot` interface should be deprecated and eventually removed.

Alternatives

Continue mixing PXE and iPXE in single driver interface and setting iPXE *vs* PXE globally.

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

ironic CLI

None

openstack baremetal CLI

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

None

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

- A new config section `[ipxe]` will be added, with most of the ipxe_* options copied from current `[pxe]` section (with `ipxe_` removed from option names). By the virtue of `oslo_config` library, the new options will be backward compatible with old, deprecated ones, using their values when not set explicitly.
- This change has no immediate effect. Enabling it is a conscious choice of the operator:
  - a driver utilizing this new `iPXEBoot` boot interface must be composed
  - such driver must be assigned to the node

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
pshchelo (Pavlo Shchelokovskyy)

Work Items

- create a new `iPXEBoot` boot interface
  - identify and refactor code that is common for `PXEBoot` and `iPXEBoot` interfaces
- register this driver as entry point for `ironic.hardware.interfaces.boot`

  - add it to list of boot interfaces enabled by default in ironic config (`[DEFAULT]enabled_boot_interfaces` config option)

- add appropriate documentation

Dependencies

- Dynamic iPXE configuration
- Driver composition reform

Testing

No specific coverage seems to be needed apart from enabling a driver that uses this new proposed boot interface at least on one gate job.

Upgrades and Backwards Compatibility

The feature has no immediate effect on existing installation as it must be manually enabled first by enabling the interface and composing an appropriate driver with this boot interface.

Existing drivers do not depend on this feature in any way.

It is also proposed to deprecate and eventually remove iPXE capabilities from the PXEBoot interface.

Chain-loading an iPXE-capable boot-loader will still be supported by iPXEBoot driver to support older/dumber/buggy hardware.

Documentation Impact

Document new driver interface and its usage.

References

Out of Band Inspection support for `redfish` hardware type

https://bugs.launchpad.net/ironic/+bug/1668487

This proposal adds the ability to inspect/update hardware properties and auto-create ports in OOB manner for Redfish hardware type.

Problem description

Node inspection automatically collects and updates node properties. These properties can be used to seggregate bare metal nodes into appropriate resource classes that could be used in nova scheduling. Node inspection also creates ironic ports for all discovered NIC(s). The `redfish` hardware type has support for inband inspection using `inspector` (Ironic Inspector). The DMTF standard Redfish schemas supports OOB interfaces to fetch most of the inspection properties supported by ironic. By adding support for OOB inspection, the overall time needed to introspect a bare metal can be reduced.

Proposed change

This spec proposes add OOB inspection support for `redfish` hardware type. It would discover ironic supported node properties and capabilities for Redfish compliant servers. This will be done by enhancing Sushy library to fetch the required properties from Redfish controller running on the bare metal BMC.

The following mandatory properties will be discovered and updated in `node.properties` for `redfish` hardware type, as discussed in Introspect spec

- memory size
- CPUs
- CPU architecture
- NIC(s) MAC address
- disks

It would also implement the additional capabilities discussed in Common Ironic Capabilities spec and available using DMTF standard Redfish schemas for `redfish` hardware type.

The properties which are already set will be overridden upon invocation of `inspect_hardware()` except for ironic ports. If a ironic port already exists, it will not create a new port for that MAC address. It will take care of adding as well as deleting of the ports for NIC changes as discussed in Introspect spec. Not all the capabilities supported by ironic are available in all Redfish compliant servers. If a property is not available in the hardware, the property will not be added/updated in node.properties as capabilities.

Inspection would return failure in the following cases:

- Failed to get basic properties.
- Failed to get capabilities, due to service configuration errors.
- Communication errors with Redfish manager.

Sushy changes

Implement `InspectInterface` method `inspect_hardware` in Sushy library.

Alternatives

One can continue to discover these properties using inband mechanism of `inspector` supported by `redfish` hardware type.

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

ironic CLI

None

openstack baremetal CLI

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

None

Security impact

None

Other end user impact

With OOB inspection, time required for hardware introspection would be reduced.

Scalability impact

None

Performance Impact

None

Other deployer impact

With OOB inspection, time required for hardware introspection would be reduced.

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
stendulker

Other contributors:
agarwalnisha1980

Work Items

- Implementation of the `InspectInterface` class and its methods `inspect_hardware()`, `validate()` and `get_properties()`.
- Enhance Sushy library to discover required hardware properties.

Dependencies

- Depends on Sushy library

Testing

- Unit tests will be added conforming to ironic testing requirements.
- CI support will be added for inspection server using virtual CI based on sushy-tools.

Upgrades and Backwards Compatibility

None

Documentation Impact

Redfish hardware type document would be updated for OOB inspection feature.

References

- Introspect spec
- Common Ironic Capabilities spec
- Sushy library
- Redfish hardware type

5.7 Rocky

5.7.1 11.1

Boot from a Ramdisk

https://storyboard.openstack.org/#!/story/1753842

While Ironic, by its very operational nature supports booting from a ramdisk, we do not empower or enable operators who wish to utilize instances that boot from ramdisks.

In the use cases desired, these are often ephemeral instances, with no actual back-end storage. That being said, we of course cant forget cleaning and other mechanisms that help provide improved value for Ironic in a deployment.

Most of these cases are where an operator is sufficiently knowledgeable to construct a ramdisk and kernel image that can be explicitly booted from RAM. Such cases are common in High Performance Computing environments where instances may need to be quickly deployed, or are functionally disk-less in order to remove a point of failure.

Problem description

- Operators desire fully ephemeral instances that essentially operate in RAM only. This allows compute-local disks to be dedicated to ephemeral storage, or to be omitted from the machine entirely.
- Large scale cluster computing practices have long used prepared kernel/ramdisk images in order to help facilitate the stand-up and operation of their environments.
- Ironic has _some_ of the support required. We do something somewhat similar with boot from volume, and if we ever support RBD booting, we would functionally require this exact feature. What delineates this from the Boot from Volume iPXE chain loading scenarios and virtual media based booting from a remote volume, is that Boot from Volume scenarios are ultimately reliant upon back-end block storage, which is not always needed or desired for all cases.

Proposed change

We propose to build a `ramdisk` deploy interface built upon our existing ramdisk code and facilities. This would allow us to have a jump start on working functionality and help us better identify where refactoring must be required.

The `ramdisk` deployment driver interface would be an opt-in interface which focuses on supporting ramdisk booting cases.

The `ramdisk` deployment driver would also loop in the agent interface classes, in order to minimize complexity and allow for cleaning workflow and related code to remain in one location.

Mechanism wise, if the `ramdisk` deployment interface is set:

- The instance can boot based upon any provided instance `kernel` and `ramdisk` as part of the requested glance image to be deployed. Effectively any image contents would be ignored.
- The instance `boot_option` will be ignored by the ramdisk interface, and will be explicitly set to `ramdisk`. If otherwise set, the interface logs a warning.
- The same `kernel` and `ramdisk` parameters that are used for a glance image based deployment could be re-used to contain a URL.
  - In the future, ironic should consider extending the accepted URL format to include `nfs://${server_ip}{$path}{$file}`, which would enable direct boot to NFS hosted kernel and ramdisk. This would not consist of `nfsroot` scenarios. This document does not require that is performed, but the Boot from Volume specification does explicitly state that could be a scenario implemented.
  - An additional consideration could also be to support direct specification of a path to chainload to.

- Users must explicitly allow TFTP and/or HTTP iPXE endpoint access for booting the baremetal nodes. Typically deployments do not allow tenant networks to access to these endpoints.
- Configuration drives would be ignored by ironic, and users would be advised to make use of the metadata service. Deployments with a configuration drive will explicitly result in an warning being logged by the conductor.
  - Operators who directly deploy nodes using ironic may need to pass additional kernel command line arguments to the node being deployed via a ramdisk. In this case, a `/instance_info/ramdisk_kernel_arguments` field will be accepted to allow those operators to pass information to the instance.

Building this functionality would allow us to also quickly adapt the support to enable Ceph RBD based booting as well as ramdisk booting to an NFS share serving as the root device. Both are features some operators have expressed desire for. Ceph RBD support would need to be in the underlying `pxe` boot interface, as the ramdisk interface overlays the `pxe` interface. Support for Ceph RBD or NFS based booting, from `volume target` information, would be a future enhancement.

Alternatives

We could consider building logic to handle and support this into the standard deployment workflow, however given this is a sufficient and specialized use case, that it might be completely unnecessary as it would not be used in most cases.

Data model impact

None.

State Machine Impact

None. Deployment would consist of setting the boot configuration and then powering on the network node.

REST API impact

None

Client (CLI) impact

ironic CLI

None

openstack baremetal CLI

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

None

Security impact

No additional security impact is expected aside from a deployment scenario being able to be setup by an operator where it may be necessary to keep the deployed kernel/ramdisk potentially for an elongated period of time.

Ironic already does this with instances that are netboot.

Other end user impact

Operators which choose to use this feature may wish to put in place specialized network controls to facilitate the machines network booting.

Each deployment and case will be different, and without post-implementation information, we will be unable to determine if there is a standard that can be derived.

Scalability impact

No scalability impact anticipated.

Performance Impact

No substantial performance impact anticipated, although if the feature gains popularity takeover naturally takes longer.

Other deployer impact

None

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
Julia Kreger (TheJulia)

Other contributors:
None

Work Items

- Create deploy interface
- Create tempest test for said ramdisk deploy interface.
- Create user documentation.

Dependencies

Testing

This seems like a feature that could be easily tested via a tempest scenario if the driver is available. No additional testing should be required.

Upgrades and Backwards Compatibility

None

Documentation Impact

Documentation will need to be updated to support this effort.

References

None

Conductor/node grouping affinity

https://storyboard.openstack.org/#!/story/2001795

This spec proposes adding a `conductor_group` property to nodes, which can be matched to one or more conductors configured with a matching `conductor_group` configuration option, to restrict control of those nodes to these conductors.

Problem description

Today, there is no way to control the conductor-to-node mapping. This is desirable for a number of reasons:

- An operator may have an Ironic deployment that spans multiple sites. Without control of this mapping, images may be pulled over WAN links. This causes slower deployments and may be less secure.
- Similarly, an operator may want to map nodes to conductors that are physically closer to the nodes in the same site, to reduce the number of network hops between the node and the conductor. A prime example of this would be to place a conductor in each rack, reducing the path to only go through the top-of-rack switch.
- A deployer may have multiple networks for out-of-band control, that must be completely isolated. This feature would allow isolating a conductor to a single out-of-band network.
- A deployer may have multiple physical networks that not all conductors are connected to. By configuring the mapping correctly, conductors can manage only the nodes which they can communicate with. This is described further in another RFE.[0]

Proposed change

We propose adding a `conductor_group` configuration option for the conductor, which is a single arbitrary string specifying some grouping characteristic of the conductor.

We also propose a `conductor_group` field in the node record, which will be used to map a node to a conductor. This matching will be done case-insensitive, to make things a bit easier for operators.

A blank `conductor_group` field or config is the default. A conductor without a group can only manage nodes without a group, and a node without a group can only be managed by a conductor without a group.

The hash ring will need to be modified to take grouping into account, as described below in the *RPC API Impact* section.

Alternatives

Another RFE[1] proposes a complex system of hard and soft affinity, affinity and anti-affinity, and scoring of placement to a conductor with multiple tags. This is quite complex, and I dont believe well get it done in the short term. Completing this more basic work doesnt block this more complex work, and so we should take it one step at a time.

Data model impact

A `conductor_group` field will be added to the nodes table, as a `VARCHAR(255)`. This will have a default of `""`, or the empty string. This string will be used in the hash ring calculation, so theres no sense in defaulting to `NULL`.

A `conductor_group` field will also be added to the conductors table, also as a `VARCHAR(255)`. This will also have a default of `""`, or the empty string. This will be used to build the hash ring to look up where nodes should land.

State Machine Impact

None.

REST API impact

The `conductor_group` field of the node will be added to the node object in the REST API, with a microversion as usual. It will be allowed in POST and PATCH requests. As with the database, it will be restricted to 255 characters. There must be a conductor in that group available, as the conductor services node creation and updates, and is selected via the hash ring.

Its worth noting that we would like to expose the grouping of conductors via the REST API eventually. However, the best way to do this isnt immediately clear, so we leave it outside the scope of this spec for now. Another RFE[3] proposes a service management API that may be a good fit.

Client (CLI) impact

ironic CLI

None, its deprecated.

openstack baremetal CLI

The `conductor_group` field for a node will be exposed in the client output, and added to the `node create` and `node set` commands.

RPC API impact

This will affect which conductor is the destination for RPC calls corresponding to a given node, however wont have a direct effect on the RPC API itself.

The hash ring will change such that the internal keys for the hash ring will now be of the structure `"$conductor_group:$drivername"`. A colon (`:`) is used as the separator between the two, to eliminate conflicts between conductor groups and drivers or hardware types. For example, an `agent_ilo` key with no separator could mean a node with no group and the `agent_ilo` driver, or it could mean a node with group `agent_` using the `ilo` hardware type. To handle upgrades, hash ring keys will be built without the conductor group while the service is pinned to a version before this feature, and built with the conductor group when the service is unpinned or pinned to a version after this feature is implemented.

We handle upgrades by ignoring grouping for services which have a pin in the RPC version that is less than the release with this feature. Once everything is upgraded and unpinned, we begin using the grouping tags configured.

Operators should leave a sufficient number of conductors available without a grouping tag configured to run the cluster, until nodes can be configured with the grouping tag. Any nodes without a grouping tag will only be managed by conductors without a grouping tag.

Driver API impact

Hash ring generation and lookup will include the grouping tag, as specified above in the *RPC API Impact* section.

Nova driver impact

This change is transparent to Nova.

Ramdisk impact

None.

Security impact

No direct impact; however this provides another mechanism for securing a deployment by enabling logical infrastructure segregation.

Other end user impact

None.

Scalability impact

None.

Performance Impact

None.

Other deployer impact

Deployers that wish to use this feature will need to manage the process of labeling conductors and nodes to enable it, which may be a non-trivial task.

Developer impact

None.

Implementation

Assignee(s)

Primary assignee:
jroll

Other contributors:
dtantsur

Work Items

- Add database fields.
- Add conductor config and populate conductor DB field.
- Change the hash ring calculation, and bump the RPC API so that we can pin during upgrades.
- Add fields to the node and conductor objects.
- Make the REST API changes.
- Update the client library/CLI.
- Document the feature.

Dependencies

None.

Testing

Unit tests should be sufficient, as thats how we test our hash ring now. Its difficult to test this with Tempest without exposing conductor grouping via the REST API.

Upgrades and Backwards Compatibility

This is described in the *RPC API Impact* section.

Documentation Impact

This should be documented in the install guide and admin guide.

References

[0] https://storyboard.openstack.org/#!/story/1734876

[1] https://storyboard.openstack.org/#!/story/1739426

[2] Notes from the Rocky PTG session:
https://etherpad.openstack.org/p/ironic-rocky-ptg-location-awareness

[3] https://storyboard.openstack.org/#!/story/1526759

Deployment Steps Framework

https://storyboard.openstack.org/#!/story/1753128

There is a desire for ironic to support customizable and extendable deployment steps, which would provide the ability to prepare bare metal nodes (servers) that better match the needs of the users who will be using the nodes.

In order to support that, we propose refactoring the existing deployment code in ironic into a deployment steps framework, similar to the cleaning steps framework.

Problem description

Presently, ironic provides a way to prepare nodes prior to them being made available for deployment (see state diagram). This is done via cleaning. However, it is not always possible, efficient, or effective to perform some of these preparations without knowing the requirements of the users of the nodes. In addition, there may be operations that should only be done once the users requirements are known.

For example, during cleaning, a node could be configured for RAID. However, this might not be the desired RAID configuration that the user of the node wants. Since the users desires are only known at deployment time, a mechanism that allows for custom RAID configuration during deployment is preferred.

Features like custom RAID configuration, BIOS configuration, and custom kernel boot parameters are a few use cases that would benefit from a way of defining deployment steps at deploy time, in ironic.

It makes sense to provide support for this via deployment steps. This would be conceptually similar to the cleaning steps supported by ironic already.

Proposed change

This proposal is the first step in providing support for performing different deployment operations based on the users desires. (The RFE to reconfigure nodes on deploy using traits is an example of a feature that depends on this work.)

The proposed change is to implement a deployment steps (or `deploy steps`) framework that is very similar to the existing framework for automated and manual cleaning. (This was discussed and agreed upon in principle, at the OpenStack Dublin PTG.)

This change is internal to ironic. Users will not be able to affect the deployment process any more than they can do today.

Conceptually, the clean steps model is a simple idea and operators are familiar with it. Having similar deploy steps provides consistency and it will be easier for operators to adopt, due to their familiarity with clean steps. It is also powerful in that, at the end of the day (or year or two), a particular step could be a clean step, a deploy step, or both.

This includes re-factoring of code to be used by both clean and deploy steps.

The existing deployment process will be implemented as a list of one (or more) deploy steps.

What is a deploy step?

Similar to clean steps, functions that are deploy steps will be decorated with `@deploy_step`, defined in ironic/drivers/base.py as follows:

```
def deploy_step(priority, argsinfo=None):
   """Decorator for deployment steps.

   :param priority: an integer priority; used for determining the order in
       which the step is run in the deployment process. (See below,
       "When are deploy steps executed" for more details.)
   :param argsinfo: a dictionary of keyword arguments where key is the name of
       the argument and value is a dictionary as follows:

           description: <description>. Required. This should include
                        possible values.
           required: Boolean. Optional; default is False. True if this
                       argument is required.
```

An alternative is to have one decorator that allows specifying a function to be a clean step and/or a deploy step, e.g.:

```
@step(clean_priority=0, deploy_priority=0, argsinfo=None)
```

However, clean steps are abortable and deploy steps arent (yet, see below), and it is unclear whether other arguments might be added for the deploy step decorator. Thus, it seems safer and simpler to have a separate decorator for deploy steps. (Having one decorator for both types of steps is left as a future exercise.)

Although ironic allows cleaning to be aborted, ironic doesnt allow the deployment to be aborted (although there is an RFE to support abort in deploy_wait). So it is outside the scope of this specification.

A deploy step can be implemented by any Interface, not just DeployInterface.

When are deploy steps executed?

Each deploy step has a priority; a non-negative integer. In this first phase, the priorities will be hard-coded. There will be no way to turn off or change these priorities.

The steps are executed from highest priority to lowest priority. Steps with priorities of zero (0) are not executed. A step has to be finished, before the next one is started.

Alternatives

There may be other ways to provide support for customizable deployment steps per user/instance, but there doesnt seem to be good reasons for having a different design from that used for clean steps.

We could choose not to provide support for customized deploy steps on a per user/instance basis. In that case, some of the current workarounds to overcome this problem include:

- have groups of nodes configured in advance (using clean steps) for each required combination of configurations. This could lead to strange capacity planning issues.
- executing the desired configuration steps after each node is deployed. As these configuration steps are executed post-deploy, most of them need a reboot of the node, orchestration is needed to do these reboots properly, and this causes performance issues that are not acceptable in a production environment. This approach wont work for pre-deploy steps though, such as RAID for the boot disk.
- users can create their own images for each use case. But the limitation is that the number of images can grow exponentially, and that there is no ability to match a specific type of hardware with a specific image.
- use a customizable DeployInterface like the ansible deploy interface (although the ansible deploy interface is not recommended for production use). This may not be able to achieve the same level of access to the hardware or settings, to have the same effect.

Data model impact

Similar to clean steps, a Node object will be updated with:

- a new `deploy_step` field: this is the current deploy step that is being executed or None if no steps have been executed yet. This will require an update to the DB.
- `driver_internal_info['deploy_steps']`: the list of deploy steps to be executed.
- `driver_internal_info['deploy_step_index']`: the index into the list of deploy steps (or None if no steps have been executed yet); this corresponds to node.deploy_step.

State Machine Impact

No new state or transition will be added.

The state of the node will alternate from states.DEPLOYING (`deploying`) to states.DEPLOYWAIT (`wait call-back`) for each asynchronous deploy step.

REST API impact

There will not be any new API methods.

GET /v1/nodes/*

The GET /v1/nodes/* requests that return information about nodes will be modified to also return the nodes `deploy_step` field and the deploy-related information in the nodes `driver_internal_info` field.

Similar to the `clean_step` field, the `deploy_step` field will be the current deploy step being executed, or None if there is no deployment in progress (or hasnt started yet).

If the deployment fails, the `deploy_step` field will show which step caused the deployment to fail.

This change requires a new API version. For nodes that have not yet been deployed using the deploy steps, the `deploy_step` field will be None, and there wont be any deploy-related entries in the `driver_internal_info` field.

For older API versions, this `deploy_step` field will not be available, although any deploy-related entries in the `driver_internal_info` field will be shown.

Client (CLI) impact

The only change (when the new API version is specified), is that the response for a Node will include the new `deploy_step` field and during deployment, the new deploy-step-related entries in the nodes `driver_internal_info` field.

ironic CLI

Even though this has been deprecated, responses will include the change described above.

openstack baremetal CLI

Responses will include the change described above.

RPC API impact

None.

Driver API impact

Similar to cleaning, these methods will be added to the drivers.base.BaseInterface class:

```
def get_deploy_steps(self, task):
    """Get a list of deploy steps this interface can perform on a node.

    :param task: a TaskManager object, useful for interfaces overriding this␣
↪method
    :returns: a list of deploy step dictionaries
    """

def execute_deploy_step(self, task, step):
    """Execute the deploy step on task.node.

    :param task: a TaskManager object
    :param step: The dictionary representing the step to execute
    :raises DeployStepFailed: if the step fails
    :returns: None if this method has completed synchronously, or
        states.DEPLOYWAIT if the step will continue to execute
        asynchronously.
    """
```

The actual deploy steps will be determined in the coding phase; we will start with one big deploy step (to get the framework in) and then break that step up into more steps determined by what makes sense

given the existing code, and the constraints (e.g. support for out-of-tree drivers, backwards compatibility when a deploy step in release N is split into several steps in release N+1).

(This specification will be updated with the actual deploy steps, once that is determined.)

Out-of-tree Interfaces

Although the conductor will still support deployment the old way (without deploy steps), this support will be deprecated and removed based on the standard deprecation policy. (The deprecation period may be extended if there is a strong desire to do so by the vendors; were flexible.)

For out-of-tree interfaces that dont have deploy steps, the conductor will emit (log) a deprecation warning, that the out-of-tree interface should be updated to use deploy steps, and that all nodes that are being deployed using the old way, need to be finished deploying, before an upgrade to the release where there is no longer any more support for the old way.

Nova driver impact

None

Ramdisk impact

There should be no impact to the ramdisk (IPA).

In the future, when we allow configuration and specification of deploy steps per node, we might provide support for collecting deploy steps from the ramdisk, but that is out of scope for this first phase.

Security impact

None

Other end user impact

None.

Scalability impact

None.

Performance Impact

None.

Other deployer impact

None.

Developer impact

DeployInterfaces (and any other interfaces involved in the deployment process) will need to be written with deploy steps in mind.

Implementation

Assignee(s)

Primary assignee:

- rloo (Ruby Loo)

Work Items

Ironic:

- Add support for deploy steps to base driver
- rework the existing code into one or more deploy steps
- Update the conductor to get the deploy steps and execute them

`python-ironicclient`:

- Add support for node.deploy_step

Dependencies

None.

Testing

- unit tests for all new code and changed behaviour
- CI jobs already test the deployment process; they should continue to work with these changes

Upgrades and Backwards Compatibility

- Old Interfaces will work with the new BaseInterface class because the code will cleanly fall back when an Interface does not support `get_deploy_steps()`. A deprecation warning will be logged, and we will remove support for the old way according to the OpenStack policy for deprecations & removals.
- Likewise, an Interface implementation with `get_deploy_steps()` will work in an older version of Ironic.
- In a cold upgrade:
  - if the agent heartbeats and driver_internal_info[deploy_steps] is empty, proceed the old way.
  - if a deployment is started by a conductor using deploy steps (new code), it means all the conductors are using the new code, so the deployment can continue on any conductor that supports the node
- In a rolling upgrade:
  - if the agent heartbeats and driver_internal_info[deploy_steps] is empty, proceed the old way (similar to cold upgrade)
  - a new conductor will not use the deploy steps mechanism if it is pinned to the old release (via *pin_release_version* configuration option). if a deployment is started by a conductor using deploy steps (new code), it means that it is unpinned, and all the conductors are using the new code, so the deployment can continue on any conductor that supports the node.

Documentation Impact

- api-ref: https://developer.openstack.org/api-ref/baremetal/ will be updated to include the new node.deploy_step field

References

- cleaning
- OpenStack Dublin PTG etherpad
- RFE to reconfigure nodes on deploy using traits
- RFE to support abort in deploy_wait
- state diagram

Hardware interface for BIOS configuration

https://bugs.launchpad.net/ironic/+bug/1712032

The proposal is intended to create a new hardware interface for BIOS automated configuration and a method to make BIOS configuration available as part of manual cleaning[0].

Problem description

- There are several use cases that need to configure BIOS options to enable certain functionality or gain performance optimization on OpenStack baremetal nodes. For example, in order to use SRIOV[1] or DPDK[2] technologies, the Virtualization Technology BIOS option shall be enabled; to achieve deterministic low packet latency in real time scenario, BIOS options related to Power Management, CPU sleep states etc shall be disabled; another good example is console redirection BIOS settings.

Proposed change

- After a node is enrolled and the basic hardware information is available, the operator can define a BIOS configuration.
- The operator will be able to set the BIOS configuration for a specific node using a cleaning step which will be defined on the new `BIOSInterface`. The BIOS settings to be changed will be passed in the form of a JSON dictionary as an argument to the cleaning step. This cleaning step will be manual at the moment, but it may be extended to be an automated cleaning step in the future.
- Similar to the RAIDInterface, this proposes a new hardware interface called `BIOSInterface` which will be used for out-of-band BIOS configuration for hardware types. Calling it BIOSInterface doesnt mean the spec is specific to BIOS systems, but a general interface name for systems such as BIOS, UEFI etc. Please refer to Driver API impact for a list of methods that will be added for this interface.

[0] Manual cleaning - https://github.com/openstack/ironic-specs/blob/master/specs/approved/manual-cleaning.rst
[1] SRIOV BIOS settings - https://docs.openstack.org/neutron/latest/admin/config-sriov.html#create-virtual-functions-compute
[2] DPDK BIOS settings - http://dpdk.org/doc/guides/linux_gsg/sys_reqs.html

> **Note**
>
> The reason of using BIOS instead of others is that its hard to find a proper name that applies to all the systems and BIOS is agreed to be close enough.

- The credentials of BIOS configuration which reuse the existing credentials in `driver_info` will be validated when going from enroll to manageable.
- A new database table `bios_settings` will be created if not exist and the current BIOS settings will be retrieved and updated for the node as part of entering cleaning step in `_do_node_clean`, this means it will be called for both manual and automated cleaning. After the given BIOS options are successfully applied in baremetal, it will update the cached bios settings with applied changes.
- This interface will not be available with any of the classic drivers. And classic drivers would throw UnsupportedDriverExtension for this interface.
- If there is no BIOS interface for a node (i.e. bios_interface=no-bios), an attempt to change BIOS configuration will result in cleaning step not found error.

Alternatives

- Operator can change the BIOS configuration manually whenever required. But this has to be done for each node which is time-consuming and error-prone.

Data model impact

- Unlike the RAID configuration[3], the target BIOS settings will be passed as an argument to cleaning step and not stored in the database.
- The current BIOS config will be cached and will be stored in a separate BIOS table. The following database table and fields will be added:
  - A new table named `bios_settings` will be added with the following fields:
    - node_id
      - Integer
      - PrimaryKeyConstraint
      - ForeignKeyConstraint(nodes.id)
    - name
      - String
      - PrimaryKeyConstraint
    - value
      - String
    - created_at
      - DateTime

[3] RAID configuration - https://github.com/openstack/ironic-specs/blob/master/specs/approved/ironic-generic-raid-interface.rst

    * updated_at
        - DateTime

    It will store the cached BIOS information that was retrieved from the node and will be updated when the BIOS settings are changed. created_at and updated_at fields will be updated accordingly when a new record is added or the existing record is updated.

- `node.bios_interface` will be added to node table, and it will contain the hardware interface we want to use for BIOS automation.
- New objects `ironic.objects.bios.BIOSSetting` and `ironic.objects.bios.BIOSSettingList` will be added to object model. The `BIOSSetting` and `BIOSSettingList` fields in the python object model will be populated on-demand.

State Machine Impact

- When going from enroll to manageable, credentials are validated to make sure user has the proper rights to access the BIOS config.

REST API impact

Two new cleaning steps are proposed to be implemented on the `BIOSInterface`:

- bios.factory_reset. It will trigger the BIOS settings factory reset for a given node. For example:

```
{
  "target":"clean",
  "clean_steps": [{
    "interface": "bios",
    "step": "factory_reset"
  }]
}
```

- bios.apply_configuration. It will set the given settings to the BIOS of a given node. For example:

```
{
  "target":"clean",
  "clean_steps": [{
    "interface": "bios",
    "step": "apply_configuration",
    "args": {
      "settings": [
        {
          "name": <name>,
          "value": <value>
        },
        {
          "name": <name>,
          "value": <value>
        }
      ]
    }
```

(continues on next page)

(continued from previous page)

```
  }]
}
```

- A new REST API will be introduced to get the cached BIOS config for a node:

```
GET /v1/nodes/<node_ident>/bios
```

The operation will return the currently cached settings with the following data schema:

```
{
  "bios": [
    {
      "links": [
        {
          "href": "http://127.0.0.1:6385/v1/nodes/<node_ident>/bios/<name>
↪",
          "rel": "self"
        },
        {
          "href": "http://127.0.0.1:6385/nodes/<node_ident>/bios/<name>",
          "rel": "bookmark"
        }
      ],
      "name": <name>,
      "value": <value>
    },
    {
      "links": [
        {
          "href": "http://127.0.0.1:6385/v1/nodes/<node_ident>/bios/<name>
↪",
          "rel": "self"
        },
        {
          "href": "http://127.0.0.1:6385/nodes/<node_ident>/bios/<name>",
          "rel": "bookmark"
        }
      ],
      "name": <name>,
      "value": <value>
    }
  ]
}
```

The API will return HTTP 400 (Bad Request) if driver doesnt support BIOS configuration or 404 (Resource Not Found) if node BIOS has not yet been configured. Otherwise it will return HTTP 200 (OK).

- To get a specified BIOS setting for a node:

```
GET /v1/nodes/<node_ident>/bios/<setting name>
```

The operation will return the specified BIOS setting with the following data schema:

```
{
  "<setting name>":
    {
      "name": <setting name>,
      "value": <value>
    }
}
```

Client (CLI) impact

ironic CLI

The ironic CLI will not be updated.

openstack baremetal CLI

- To retrieve the cached BIOS configuration with node-uuid:

```
$ openstack baremetal node bios setting list <node-uuid>
```

- To show a specified BIOS setting with node-uuid:

```
$ openstack baremetal node bios setting show <node-uuid> <setting-name>
```

- The validation result of BIOS Interface will be returned through the standard validation interface.

RPC API impact

None

Driver API impact

A new `BIOSInterface` will be available for the drivers to allow them to implement BIOS configuration. There will be several new methods and cleaning steps in the interface:

- `do_factory_reset()` - This method is called to reset all the BIOS settings supported by driver to factory default. It will also update the records of `bios_settings` database table to the known defaults once reset action succeeds. It is up to the vendor to decide the BIOS defaults settings that will be set.
- `factory_reset()` - This cleaning step will delegate the actual reset work into the abstract method `do_factory_reset()`.

  The operator can choose to call it as part of manual cleaning steps. The corresponding manual cleaning step will be `bios.factory_reset`.
- `do_apply_configuration(configuration={})` - The driver implementation of this method will take the settings from the configuration dictionary and will apply BIOS configuration on the bare metal. The driver is responsible for doing the corresponding validation before applying the

settings, and/or manage failures when setting an invalid BIOS config. Implementation of this method needs to rollback previous settings upon first failure. In the case of needing password to update the BIOS config, it will be taken from the `driver_info` properties. The implementation detail is up to the driver.

- `apply_configuration(configuration={})` - This cleaning step will delegate the actual configuration work into the abstract method `do_apply_configuration(configuration={})`.

  The operator can choose to call it as part of manual cleaning steps. The corresponding manual cleaning step will be `bios.apply_configuration`.

- `cache_bios_settings()` - This method will be called to update BIOS configuration in `bios_settings` database table. It will attempt to get the current BIOS settings and store them in the `bios_settings` database table. It will also update the timestamp fields of created_at and updated_at accordingly. The implementation detail is up to the driver, for example, whether to have a sub method shared by `do_factory_reset`, `do_apply_configuration` and `cache_bios_settings` to retrieve and save bios information in `bios_settings` table.

Nova driver impact

None

Ramdisk impact

None

Security impact

Unprivileged access to the BIOS configuration can expose sensitive BIOS information and configurable BIOS options to attackers, which may lead to disruptive consequence. Its recommended that this kind of ability is only restricted to administrative roles. Changing BIOS settings requires credentials which will reuse the existing credentials in `driver_info` instead of creating new fields.

Other end user impact

None

Scalability impact

None

Performance Impact

BIOS configuration may extend the time required for manual cleaning on the nodes.

Other deployer impact

- Add new config options:
  - `enabled_bios_interfaces`: a list of enabled bios interfaces.
  - `default_bios_interface`: default bios interface to be used.
- Operator can use `bios.apply_configuration` and `bios.factory_reset` as manual cleaning tasks for doing BIOS management.

Developer impact

Developer may implement the `BIOSInterface` for respective drivers.

Implementation

Assignee(s)

Primary assignee:
zshi yroblamo

Work Items

- Add bios interface field in Node object.
- Create database model & api for nodes bios table and operations.
- Create `BIOSInterface` which includes the following items:
  - Add new methods in `BIOSInterface` base driver, such as `do_apply_configuration`, `do_factory_reset` and `cache_bios_settings`.
  - Add new cleaning steps in `BIOSInterface` base driver, such as `apply_configuration`, `factory_reset`.
  - Add caching of BIOS config as part of entering cleaning step in `_do_node_clean`.
- Create fake & no-bios implementation derived from `BIOSInterface`.
- Create REST API endpoints for BIOS configuration.
- Create RPC objects for BIOS configuration.
- Implement OSC baremetal CLI changes.

Dependencies

None

Testing

- Unit tests will be added for the code. A fake implementation of the `BIOSInterface` will be provided with `do_apply_configuration` method for testing purposes and this can be run as part of manual cleaning.
- Each driver is responsible for providing the third party CI for testing the BIOS configuration.
- Tempest tests will be added using fake driver.

Upgrades and Backwards Compatibility

- Raise errors when there is no BIOSInterface support in driver.

Documentation Impact

- Documentation will be provided on how to configure a node for BIOS.
- API reference will be updated.
- Respective vendors should document the default BIOS values for reference.

References

5.7.2 11.0

Future of classic drivers

https://bugs.launchpad.net/ironic/+bug/1690185

This specification discusses the future of the classic drivers after the *Driver composition reform* was implemented in the Ocata cycle. Terminology here follows one of that specification.

Problem description

We do not want to maintain two approaches to building drivers long-term. It increases complexity of the source code (see e.g. driver_factory.py) and the amount of testing for 3rdparty CI.

Proposed change

The change covers several cycles:

Pike

In the Pike cycle hardware types and classic drivers will co-exist as equally supported ways of writing drivers.

- Dynamic drivers and all related API are considered stable and ready for production use.

  > **Note**
  >
  > In the Ocata release notes we called the related API additions experimental.

- No more classic drivers are accepted in tree.
- No new interfaces are added to the existing classic drivers.
- No interface implementations are changed in the existing classic drivers.
- We *recommend* the vendors to provide hardware types analogous to their existing classic drivers. 3rd party CI should provide the complete coverage of all supported boot, deploy, management and power interface combinations. Its up to the vendors to decide whether to use classic drivers or hardware types to achieve that.

Queens

In the Queens cycle we will deprecate classic drivers.

- We will *require* the vendors to provide hardware types analogous to their existing classic drivers. It is up to the vendors to choose the combination of interfaces to support. It will be *recommended*, however, to keep support for standard deploy and inspect interface implementations, if possible.

- 3rd party CI will have to cover all hardware types and all supported combinations of the boot, deploy, management and power interfaces. 3rd party CI will be able to stop covering supported classic drivers, when their functionality is covered through hardware types.
- The classic drivers mechanism will be deprecated, and loading any classic driver (in-tree or out-of-tree) will result in a deprecation warning. The `enable_drivers` configuration option will be also deprecated.

> **Note**
>
> In the Queens release we will continue running regular CI against classic drivers still.

- Existing (in-tree) classic drivers will only receive critical bug fixes as related to the classic interface (i.e. they will still be affected by fixes in the interface implementations they share with hardware types).
- Most of the upstream CI will run on the dynamic drivers (`ipmi`, `snmp` and `redfish`). The standalone job will provide coverage for classic drivers. Grenade will be testing switch from classic drivers (e.g. `pxe_ipmitool`) to hardware types (e.g. `ipmi`).
- Deprecate `-t/--type` argument to driver listing commands in `python-ironicclient`.

> **Note**
>
> Deprecating or removing the `type` argument to the driver listing API is outside of the scope of this proposal.

- Extend the upgrade documentation to contain a full mapping between supported classic drivers and associated combination of a hardware type and hardware interfaces. Explicitly mention classic drivers that will not receive a new counterpart per vendor decision, and which replacement is recommended for such drivers.
- Update the whole documentation to only mention hardware types, except for the driver-specific documentation and the upgrade documentation bit explained above.
- Provide automatic migration to hardware types as part of the `online_data_migration` command - see *Automatic migration*.

> **Note**
>
> We decided to not provide any automatic migration on the API level in the node create and update API. Doing so would require us to maintain mapping between classic drivers and corresponding hardware types/interfaces foreever. It also may be confusing for operators, if, for example, the result of the node creation request differs from the outcome.

Rocky

In the Rocky release the support for classic drivers is removed.

- Remove all in-tree classic drivers.
- Remove support for loading classic drivers from driver_factory.py.

- Remove the `enable_drivers` configuration option.
- Remove CI coverage for classic drivers.
- Remove `-t/--type` argument to driver listing commands in `python-ironicclient`.
- Update the driver listing API to always return an empty result when `classic` type is requested.

Automatic migration

To simplify transition for operators, make `online_data_migration` in the Queens release automatically update nodes.

- Extend BaseDriver with a new class method:

```
@classmethod
def to_hardware_type(cls):
    """Return corresponding hardware type and hardware interfaces.

    :returns: a tuple with two items:

        * new driver field - the target hardware type
        * dictionary containing interfaces to update, e.g.
          {'deploy': 'iscsi', 'power': 'ipmitool'}
    """
```

  For example, for the `agent_ipmitool` driver:

```
@classmethod
def to_hardware_type(cls):
    if CONF.inspector.enabled:
        inspect_interface = 'inspector'
    else:
        inspect_interface = 'no-inspect'

    return 'ipmi', {'boot': 'pxe',
                    'deploy': 'direct',
                    'inspect': inspect_interface,
                    'management': 'ipmitool',
                    'power': 'ipmitool',
                    'raid': 'agent'}
```

- Update the `online_data_migrations` to accept options for migrations in the form of `--option <MIGRATION NAME><KEY>=<VALUE>`. They will be passed as keyword arguments to the migration matching the provided name.
- Update the `online_data_migrations` command with a new migration `migrate_to_harware_types`. It will accept one option `reset_unsupported_interfaces`, which is a boolean value with the default of `False`. The migration will do the following:

  1. Load classes for all classic drivers in the `ironic.drivers` entrypoint (but do not instantiate them).
  2. For each classic driver:
     1. Calculate required changes using `DriverClass.to_hardware_type`.

Missing interfaces, other than `boot`, `deploy`, `management` and `power`, are defaulted to their no-op implementations (`no-***`).

> **Note**
>
> We consider `boot`, `deploy`, `management` and `power` mandatory, as they do not have a no-op implementation.

2. If the hardware type is not in `enabled_hardware_types`, issue a and skip all nodes with this classic driver.
3. If any interface is not enabled (not in `enabled_***_interfaces`):
    1. if this interface is one of `boot`, `deploy`, `management` or `power`, or if `reset_unsupported_interfaces` is `False`, issue a warning and skip the nodes.
    2. otherwise try again with resetting the interface to its no-op implementation (`no-***`).
4. Update the node record in the database.

> **Note**
>
> Due to idempotency of the migrations, operators will be able to re-run this command after fixing the warnings to update the skipped nodes.

- In the **Rocky** cycle, update the `dbsync` command with a check that no nodes are using classic drivers. As the list of classic drivers will not be available at that time (they will be removed from the tree), maintain the list of classic driver names that used to be in tree and check nodes against this list. Remove this check in the release after Rocky.

Alternatives

- Keep classic drivers forever. Complicates maintenance for unclear reasons.
- Start deprecation in the Pike cycle. We wanted to have at least one cycle where hardware types are fully supported before we jump into deprecation. Also, in this case we will have to rush the vendors into creating and supporting their hardware types before end of Pike.

Data model impact

None

State Machine Impact

None

REST API impact

Due to the way we designed *Driver composition reform*, dynamic drivers look very similar to the classic drivers for API point of view.

We could deprecate the `type` argument to the driver listing API. However,

1. API deprecations are hard to communicate,
2. due to API versioning, we will still have to support it forever.

Thus, this specification does not propose deprecating anything in the API.

Client (CLI) impact

ironic CLI

Deprecate `-t` argument to the `driver-list` command in the Queens cycle and remove it in Rocky.

openstack baremetal CLI

Deprecate `--type` argument to the `baremetal driver list` command in the Queens cycle and remove it in Rocky.

RPC API impact

None

Driver API impact

- In the Queens release, all classic drivers will behave as if they had `supported = False`.
- In the Rocky release, support for loading classic drivers will be removed. `BaseDriver` will be merged with `BareDriver`, code in driver_factory.py will be substantially simplified.

Nova driver impact

None

Ramdisk impact

None

Security impact

None

Other end user impact

Users of Ironic will have to switch their deployment to hardware types before upgrading to Rocky.

Scalability impact

None

Performance Impact

None

Other deployer impact

See *Upgrades and Backwards Compatibility*.

Developer impact

Out-of-tree classic drivers will not work with the Rocky release of Ironic.

Implementation

Assignee(s)

Primary assignee:
Dmitry Tantsur (IRC: dtantsur, LP: divius)

Work Items

See *Proposed Change* for the quite detailed breakdown.

Dependencies

None

Testing

Starting with the Queens release, our CI will mainly test hardware types.

We will modify the Grenade job testing Pike -> Queens upgrade to switch from `*_ipmitool` to `ipmi` during the upgrade.

Upgrades and Backwards Compatibility

Removing the drivers and the classic driver mechanism is going to be a breaking change and has to be communicated accordingly.

Operators will have to enable appropriate hardware types and hardware interfaces in the Queens release.

Documentation Impact

The upgrade guide will be updated to explain moving from classic drivers to hardware types with a examples and a mapping between old and new drivers.

References

Add Inspect Wait State

https://bugs.launchpad.net/ironic/+bug/1725211

The spec proposes adding a new `inspect wait` state to ironic state machine.

Problem description

The in-band inspection is an asynchronous process[1] that isnt currently handled through a waiting state. This is a discrepancy from the rest of the asynchronous **ironic** states and may comprise a problem for future features such as aborting the introspection[2] or merging **ironic** and **ironic-inspector**[3];

Proposed change

Lets therefore have a new passive state in the **ironic** state machine, the `inspect wait` state.

For asynchronous inspection like ironic inspector driver, ironic conductor will move node from `manageable` to `inspecting` state when an inspect request is issued, then the ironic conductor moves node to `inspect wait` state if `InspectInterface.inspect_hardware` returns `INSPECTING` or `INSPECTWAIT`. The behavior of returning `INSPECTING` will be deprecated and logged as a warning. After deprecation period, returning `INSPECTING` will cause node be moved to `inspect failed` state.

Add a new option `[conductor]inspect_wait_timeout` to guard the `inspect wait` state, the default value is 1800 seconds as same as `[conductor]inspect_timeout`. If the hardware inspection is timed out in the state of `inspect wait`, node will be moved from `inspect wait` to `inspect failed`.

The existing `[conductor]inspect_timeout` will be deprecated.

The `inspect wait` state will be set as an allowed state when updating ironic node, port and portgroup.

As ironic-inspector checks node provision state before starting inspection, the `inspect wait` state needs to be added to ironic-inspector as a valid state.

Alternatives

There are no alternatives to this feature.

Data model impact

None

State Machine Impact

A new unstable state `inspect wait` will be added to ironic state machine.

Following state transitions will be added:

1. `inspecting` to `inspect wait` with event `wait`.
2. `inspect wait` to `inspect failed` with event `fail`.
3. `inspect wait` to `manageable` with event `done`.

REST API impact

API microversion will be bumped to hide the new `inspect wait` state to clients with older microversion, this will be done in the `update_state_in_older_versions`.

Node related API endpoints will be affected:

[1] https://docs.openstack.org/ironic-inspector/pike/user/http-api.html#start-introspection
[2] https://review.opendev.org/#/c/482867/16/specs/approved/inspection-abort.rst
[3] https://etherpad.openstack.org/p/inspector-queens-virtual-ptg

- POST /v1/nodes
- GET /v1/nodes
- GET /v1/nodes/detail
- GET /v1/nodes/{node_ident}
- PATCH /v1/nodes/{node_ident}

For clients with older microversion, the provision state of `inspect wait` will be changed to `inspecting`, there is no other impact to API behaviors.

Client (CLI) impact

As the compatibility is handled at ironic API, CLI is not affected.

ironic CLI

None

openstack baremetal CLI

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

None

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

Add a new option `[conductor]inspect_wait_timeout` to guard timeout of state `inspecting`, defaults to 1800 seconds.

Developer impact

This feature has no impact on synchronous inspection, that includes most of OOB drivers. For in-band inspection, the new state has to be considered.

After this spec is implemented, drivers based on asynchronous inspection have to be changed accordingly, that includes in-band inspection and out-of-band inspection (if there is any).

`OneViewInspect` in the tree is implemented based on ironic inspector interface, its state transition from `inspecting` to `inspect wait` is handled by ironic inspector, but `inspect wait` state needs to be added to status checking.

Implementation

Assignee(s)

Primary assignee:
kaifeng

Other contributors:
vetrisko

Work Items

1. Add `inspect wait` state and state transitions to ironic state machine.
2. Apply state change in the `_check_status` of ironic inspector and `OneViewInspect` driver.
3. Add new option `inspect_wait_timeout`, and deprecate `inspect_timeout`.
4. Handle timeout of state `inspect wait` in the conductor periodic task `_check_inspect_timeouts`, allow updating node, port and portgroup when node is in the `inspect wait` state.
5. Handle API microversion compatibility.
6. Add `inspect wait` to ironic-inspector as a valid state.
7. Update documents, see *Documentation Impact* for details.

Dependencies

None

Testing

Unit tests will be added, API change will be covered by tempest tests.

Upgrades and Backwards Compatibility

The API backwards compatibility is guarded by microvision.

Documentation Impact

The state diagram will be automatically generated from source. Update ironic states document to address the new state, and the semantic change of current `inspecting` state.

References

Support baremetal inspection abort

https://bugs.launchpad.net/ironic/+bug/1703089

This spec aims to support aborting node inspection from ironic API. A dependency of `inspect wait` state in[1] is required for this spec to continue.

Problem description

Currently, we cant abort the process of node inspection from ironic API. When a node is not properly setup under inspection network, admins can only wait it to fail after specified timeout, or abort the inspection process from ironic inspector API/CLI (if the in-band inspect interface `inspector` is in use).

Although the inspection state will be synchronized to ironic by periodic task, its not consistent for an operation started from ironic, then stopped by inspector, furthermore, it creates a little delay of time. Node state is inconsistent between ironic and inspector until next state synchronization. The default time interval for ironic-inspector state synchronization is 60 seconds, it may vary depending on user configuration.

Proposed change

Add state transition of `inspect wait` to `inspect failed` to state machine, add support to ironic to allow the verb `abort` can be requested when node in `inspect wait` state.

Add a method named `abort` into `InspectInterface`, so that inspect interface can provide implementation to support inspection abort. The default behavior is to raise an `UnsupportedDriverExtension` exception. Implement the abort operation for `inspector` inspect interface.

When an abort operation is requested from ironic API, and the node in the state of `inspect wait`, ironic calls `abort` method from inspect interface of driver API, and moves node state to `inspect failed` if the method executed successfully.

Note that, the abort request to ironic-inspector is asynchronous, ironic will move node to `inspect failed` once the request is accepted (202), disregard if the operation at ironic-inspector is performed successfully. This reduces the design complexity for this feature by handling failure at the side of ironic-inspector.

From the point of view of ironic-inspector, every inspect request will refresh local cache for the node, it assures that node state is in sync when starting node inspection. However, inconsistent node state do

[1] https://specs.openstack.org/openstack/ironic-specs/specs/approved/inspect-wait-state.html

exist if abort request is accepted but not performed successfully at ironic-inspector. This inconsistency will be eliminated by ironic-inspector node cache clean up when timeout is reached.

Involved changes are:

- Add a method named `abort()` to base inspect interface (InspectInterface).
- Implement `abort()` for `inspector` inspect interface.
- Implement the logic for ironic handling the verb `abort` when provisioning state is `inspect wait`.

Alternatives

- Wait for `inspect fail` after specified timeout value.
- Request through ironic-inspector api or `openstack baremetal introspection abort` command. Be aware that its only viable when using ironic inspector as inspect interface. Other inspect interfaces like out-of-band inspection may have different approach to achieve the same goal, that is beyond the scope of this spec.

Data model impact

None

State Machine Impact

Add a state transition of `inspect wait` to `inspect failed` with event `abort` to ironic state machine.

REST API impact

Modify provision state API to support the transition described in this spec. API microversion will be bumped. For clients with earlier microversion, the verb `abort` is not allowed when a node is in `inspect wait` state.

- PUT /v1/nodes/{node_ident}/states/provision
  - The same JSON Schema is used to `abort` a node in `inspecting` state:

```
{
  "target": "abort"
}
```

  - For client with earlier microversion, 406 (Not Acceptable) is returned
  - For client with supported microversion
    - 202 (Accepted) is returned if request accepted
    - 400 (Bad Request) is returned if current inspect interface does not support abort

Client (CLI) impact

ironic CLI

None

openstack baremetal CLI

None

RPC API impact

None

Driver API impact

A new method `abort` will be added to `InspectInterface` in base.py, the default behavior is to raise the exception `UnsupportedDriverExtension`:

```
def abort(self, task):
    raise exception.UnsupportedDriverExtension(
        driver=task.node.inspect_interface,
        extension='abort')
```

Nova driver impact

None

Ramdisk impact

None

Security impact

None

Other end user impact

None

Scalability impact

For multiple nodes under inspection in a notable scale, it will reduce a little time costs in case of inspection retry.

Performance Impact

None

Other deployer impact

Deployers can abort hardware introspection through ironic API/CLI, besides the inspector API/CLI, for nodes using inspector as the (in-band) inspection interface.

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
kaifeng

Work Items

- Add transition of `inspect wait` to `inspect failed` via `abort`.
- Add a new method `abort()` to the base inspect interface.
- Add the abort implementation to ironic `inspector.Inspector`.
- Implement the abort logic in ironic conductor.

Dependencies

None

Testing

Tempest test will be added to test the REST API change.

Upgrades and Backwards Compatibility

API will be bumped for backward compatibility. Client requests with microversion before this feature will be treated identically.

Documentation Impact

Related documents and state machine diagram will be updated accordingly.

References

Power fault recovery

https://storyboard.openstack.org/#!/story/1596107

This spec proposes adding a new node field to reflect any faults detected by the system. For nodes put into maintenance mode due to power-sync failures, this proposes a mechanisms to try to recover the nodes from the failure.

Problem description

Currently, ironics API and data model do not differentiate between ironic setting maintenance on a node (for instance, when cleaning fails, or when the power status loop cannot contact the BMC) and an operator setting maintenance on a node (an explicit API action).

This situation makes it difficult to apply any power fault recovery mechanism.

Proposed change

A string field named `fault` will be added to ironics data model for a Node. It will be used to record any detected faults.

The field will be set to one of the following values, depending on the situation (when ironic puts the node into maintenance due to one of these):

- `power failure`: when a node is put into maintenance due to power sync failures that exceed max retries.
- `clean failure`: when a node is put into maintenance due to failure of a cleaning operation.
- `rescue abort failure`: when a node is put into maintenance due to failure of cleaning up during rescue abort.
- None: there is no fault detected for the node; this is the default value. Since the field is set to a fault when ironic puts the node into maintenance due to a fault, it gets set to None when the node is taken out of maintenance (by ironic or the user).

  For nodes that were in maintenance prior to upgrading to this release, the field will also be None, since we dont know for sure, if (or what) there had been a fault. Even if there had been, we dont know whether the operator wants the node in maintenance for other reasons as well.

A periodic task will be added to the conductor. This will operate on nodes with `node.fault == power failure`. It will try to get the nodes power state. If successful, the node will be taken out of maintenance.

An integer configuration option named `[conductor]power_failure_recovery_interval` will be added. This is the interval (number of seconds) between each run of the periodic task. The default value is 300 seconds (5 minutes). Setting it to 0 will disable power fault recovery.

Alternatives

Proposals have already been made and rejected to use a combination of maintenance being set and a given maintenance_reason or last_error to do self-healing due to inconsistency.

Specific faults support[1] is proposed but its too big and to-date, no consensus has been reached on it.

An earlier version of this spec had proposed a `maintenance_type` field for the node, instead of a `fault` field. We prefer the `fault` field since it better reflects that it is about faults. `maintenance_type` caused the field to be tied in with the existing `maintenance` and `maintenance_reason` fields. Using `fault` makes it distinct from `maintenance`; it will make it easier to move forward in the future, should we decide to provide more enhancements that are fault-related, or to separate the notion and handling of faults from maintenance (which some would argue should only be operator-initiated).

Data model impact

A string field named `fault` will be added to ironics node table.

The field will be added to `Node` object; object version will be incremented.

[1] https://review.opendev.org/#/c/334113/

State Machine Impact

None

REST API impact

The field `fault` will be added to `Node` API object, ironic API version will be bumped.

Node-related API endpoints will be affected:

- POST /v1/nodes
- GET /v1/nodes
- GET /v1/nodes/detail
- GET /v1/nodes/{node_ident}
- PATCH /v1/nodes/{node_ident}

For requests with older microversion, `node.fault` is hidden from the response. This will be done in the `hide_fields_in_newer_versions()`.

Field `fault` is read-only from the API, and can only be modified internally. Any POST/PATCH request to Node with `fault` set will get 400 (Bad Request).

There is no other impact to API behaviors.

Client (CLI) impact

ironic CLI

None

openstack baremetal CLI

The CLI will be updated to support field `fault`, guarded by ironic API microversion.

RPC API impact

None

Driver API impact

None

Nova driver impact

None, although we should update the ironic-virt driver code to also look at this new node.fault field, in addition to the `node.maintenance` field.

Ramdisk impact

None

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

If the periodic task for recovery is enabled, it will consume some resources on conductor node.

More will be consumed in an environment containing some nodes in maintenance due to power failure.

Other deployer impact

A new option `[conductor]power_failure_recovery_interval` is introduced to support power failure recovery. The default value is 300 (5 minutes), you have to set to 0 if this feature is not needed.

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
kaifeng

Other contributors:
dtantsur

Work Items

- Update db layer to include the `fault` field.
- Change places where nodes enter/leave maintenance, to set `fault` accordingly.
- Add `[conductor]power_failure_recovery_interval` option to ironic configuration, add periodic task to handle power recovery.
- API change.

Dependencies

None

Testing

The feature will be covered by unit tests, API change will be covered by tempest test.

Upgrades and Backwards Compatibility

ironic API change is guarded by microversion.

When upgrading, any nodes in maintenance will have `node.fault` set to None. This is because there is no easy/guaranteed way to determine if a node had been (previously) put into maintenance due to one of our internal faults. Furthermore, even if we could determine whether it was due to a fault, we dont know whether the operator wants to keep the node in maintenance for other reasons besides the fault.

Should the operator want to take advantage of the power fault recovery mechanism, they could take the nodes out of maintenance. If there are still issues, ironic will do its thing detect the fault and try to recover.

Documentation Impact

- Update api-reference.
- New option will be generated by config generator.

References

SIGHUP on ironic services

https://bugs.launchpad.net/ironic/+bug/1585595

It is a common practice for daemons to use SIGHUP as a signal to reconfigure a server by reloading configuration files (as mentioned in Wikipedia). This specification describes adding this feature to the ironic-conductor and ironic-api services.

Problem description

Issuing a SIGHUP signal (via, for example, `kill -s 1 <process-id>`) to an ironic (ironic-api and ironic-conductor) service causes the service to be restarted.

It is a common practice for daemons to use SIGHUP as a signal to reconfigure a server by reloading configuration files (as mentioned in Wikipedia). This specification describes how ironic will support SIGHUP, such that select configuration options can be reloaded during runtime.

This is useful, for example, during a rolling upgrade. After unpinning (resetting the `pin_release_version` configuration option), SIGHUP could be used to restart the services with the updated option value.

Another example is for operators to more easily enable or disable the debug logging without having to stop and (re)start the services.

Note that SIGHUP is not supported on Windows.

Proposed change

We will leverage code from these libraries:

- the oslo.service library provides support for services to handle this signal. We would change the ironic services so that they are launched with `restart_method='mutate'`. When the library code handles the SIGHUP signal, it gets any changes for mutable configuration options.
- The oslo.config library adds support for `mutable` configuration options. Only mutable configuration options can be reloaded. A mutable option note (Note: This option can be changed without restarting.) is added to the description of a mutable configuration option in the ironic.conf.sample file. It logs any changed mutable configuration options. It also logs a warning for any changed options that are not mutable.

With these changes, when a SIGHUP occurs, the service will reload with values from the mutable options. A warning is logged for changes to any immutable options.

Mutable configuration options

Mutable configuration options that will be available are:

- options from other libraries. To date, the only mutable options that are used by the ironic services are from the oslo.log library:
  - `[DEFAULT]/debug`
  - `[DEFAULT]/log_config_append`
- `[DEFAULT]/pin_release_version`

Other ironic configuration options can be made mutable in the future; such changes should have corresponding release notes. The belief is that most, if not all, of the configuration options should be made mutable. However, that is outside the scope of this specification which is to lay the groundwork to make this possible with a small number of options. When mentioning that ironic supports SIGHUP, operators might assume (incorrectly) that this applies to all configuration options, so we should make other configuration options available in a timely fashion.

The value of a mutable configuration option should not be cached; or at least, if it is cached, the value must be updated upon a SIGHUP occurrence.

Alternatives

Change the desired configuration option values, stop the service, and then start it again.

Data model impact

None.

State Machine Impact

None.

REST API impact

None.

Client (CLI) impact

None.

ironic CLI

None.

openstack baremetal CLI

None.

RPC API impact

None.

Driver API impact

None.

Nova driver impact

None.

Ramdisk impact

None.

Security impact

None.

Other end user impact

The operator will be able to change certain configuration options and issue a SIGHUP to have an ironic service restart using the changed option values.

Scalability impact

None.

Performance Impact

None.

Other deployer impact

Among other things, this can be used for rolling upgrades.

Developer impact

None.

Implementation

Assignee(s)

Primary assignee: rloo (Ruby Loo)

Work Items

- change our services so they are launched with `restart_method='mutate'`
- change the desired configuration options so that they are mutable
- make sure the mutable options are not cached, or if they are, make sure that they are updated appropriately with a SIGHUP occurrence

Dependencies

None.

Testing

If we stop and restart a service in the e.g. multinode grenade testing, we could change that and issue a SIGHUP instead.

Upgrades and Backwards Compatibility

Changing the value of mutable configuration options will now take effect when a SIGHUP is issued. We need this to support rolling upgrades.

Documentation Impact

The use of SIGHUP (in the context of `[DEFAULT]/pin_release_version`) will be documented as part of the rolling upgrade process.

References

- Wikipedia
- oslo.service library
- oslo.config library
- rolling upgrade

5.8 Queens

5.8.1 10.1

Promote Ansible deploy interface to ironic

https://blueprints.launchpad.net/ironic/+spec/ansible-deploy-driver

This spec presents a new deploy driver interface for provisioning nodes via Ansible playbooks.

Problem description

Sometime it may be required to treat (some) baremetal nodes more like pets than cattle, providing per-node custom logic of deploy process.

Currently such customization in Ironic is not an easy task, as developer/cloud administrator would have to do one or more of the following:

- Modify the code of Ironic deploy driver.
- Modify the code of Ironic Python Agent.
- Modify the code of ironic-lib.
- Rebuild IPA ramdisk.
- Upload new ramdisk to Glance or HTTP server(s).
- Update the nodes info with new ramdisk.
- Update deploy driver on Ironic conductors.
- Restart conductors.

This problem can be partially solved in deploy driver based on external templates for a configuration management tool.

Possible use cases and advantages:

- Custom actions with hardware at any stage of deploy process with vendors utilities, in-band or out-of-band.
- Easy replace usage of one Linux utility with other for deploy.
- Deep tuning of deploy process.
- Changing behavior of deploy process without restart the conductors.
- Long life time of deploy ramdisk - vendors utilities can be downloaded from external sources during deploy process.
- Separation of concerns - ironic manages the what part, Ansible the how part
- Allows BMs to be treated more like pets

Proposed change

This spec proposes such deploy interface based on Ansible[1] as configuration management solution. Effectively it uses ironic for its power, management and boot capabilities, and shifts the logic of deployment itself to Ansible playbooks.

Such deploy interface is already implemented and is available as part of `ironic-staging-drivers` project[2]. It can be used as possible deploy interface for `ipmi` hardware (and possibly other hardware as well). Therefore this spec is proposing promotion of the `ansible` deploy interface for inclusion in the core ironic as one of available deploy interfaces.

[1] https://www.ansible.com/
[2] https://opendev.org/x/ironic-staging-drivers/

Below is a short description of this deploy interface architecture and current capabilities. More information is available in the `ironic-staging-drivers` project documentation at[3].

Use case

This deploy interface is mostly suitable for undercloud-like or standalone ironic usage, where the operator of ironic service is usually the owner of the images to be deployed and/or the deployed instances themselves. It does however already include set of Ansible playbooks that try to mimic the capabilities of the standard `direct` deploy interface of ironic as close as possible.

This deploy interface is really useful when, during provisioning, there is a need to perform some low-level system configuration that would be hard to do in the already provisioned instance (like placing the root partition on a LVM/software RAID) or would require an extra reboot (like changing kernel parameters).

General overview

We chose Ansible for the following reason:

- open source (GPLv3 + parts as MIT), mature and popular enough, including OpenStack ecosystem itself
- written and extendable in Python, fits nicely in the OpenStack ecosystem
- configuration files are human-readable YAML
- agent-less by design, with minimal requirements on managed node, only Python and SSH server are required.

There are two ways to utilize Ansible from Pythons program: use Ansible Python API or run CLI utility with parameters. Ansible Python API currently does mandatory fork of callers process, this behaviour is not suitable for Ironic conductor (oslo.messaging does not allow forks at least). Besides, Ansible licensing choice (GPLv3) prohibits usage of Ansible Python API from ironic (Apache 2.0). Therefore `ansible-playbook` CLI utility is used.

Each action (deploy, clean) is described by a single Ansible playbook with roles, which is run as a whole during deployment, or tag-wise during cleaning. Control of cleaning steps is through Ansible tags and auxiliary clean steps file. The playbooks for actions can be set per-node, as is cleaning steps file.

The dreploy interface tries to reuse as much code from ironic as possible, and interface-wise is quite similar to the `direct` deploy interface.

Currently this interface supports two modes for continuing deployment or cleaning:

- having the deploy ramdisk calling back to ironic APIs heartbeat endpoint (default)
- polling the node until the ssh port is open as part of a playbook

We propose to remove the latter from the interface when moving it to ironic, as support for it is not tested in gates, it decreases performance due to polling, and in general makes code more complicated.

A custom Ansible callback plugin is used for logging. It can read the logging configuration from ironic configuration file, and thus emit log entries for Ansible events directly to the logging backend ironic is also using (works best with journald backend).

[3] http://ironic-staging-drivers.readthedocs.io/en/latest/drivers/ansible.html

Deploy

The interface prepares a set of parameters which are needed for deploy, including access info for the node to be deployed. It then executes the `ansible-playbook` script passing it all the collected information, with node access info being used to register the node in Ansible inventory at runtime.

Supported image types are whole-disk images and partition images with local boot option, **netboot is currently not supported**. Compressed images are downloaded to deploy ramdisk and converted to actual disk device/partition, RAW images are streamed to the target directly.

Creating a configdrive partition is supported for both whole disk and partition images, on both msdos and GPT labeled disks.

Root device hints are currently supported in their basic form only (with exact matches, without oslo.utils operators), there are patches to add full support on review. If no root device hint is provided for the node, first device returned as part of `ansible_devices` Ansible fact is used as root device to create partitions on or write the whole disk image to.

Cleaning

Cleaning procedure for Ansible deploy interface:

- Each cleaning step is a tag in the Ansible playbook file used for cleaning. Available steps, their priorities and corresponding tags are defined in a auxiliary cleaning steps configuration file.
- `get_clean_steps()` method returns a list of cleaning steps defined in mentioned configuration file.
- `prepare_cleaning()` method loads the same ramdisk as for deploying.
- `execute_clean_step()` method does synchronous execution of cleaning step via Ansible, executing only tasks with Ansible tags assigned to the cleaning step.

Default cleaning playbook provided with the interface supports `erase_devices_metadata` and `erase_devices` clean steps of `direct` deploy interface by executing shallow disk metadata cleanup and shredding of disk devices respectively, honoring priorities of those steps set in ironics configuration file.

Alternatives

Use a different deployment customization mechanism or dont support the pet-like treatment.

The short rundown of main pros and cons of current `ansible` deploy interface compared to already available and standard `direct` deploy interface:

- easier to extend for custom provision logic
- is not async
- does not support netboot

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

To support this new deploy interface the deploy ramdisk should include:

- a user with password-less sudo permissions - required
- running SSH server configured for access to this user via SSH keys - required
- Python interpreter (Python2 >= 2.6 or Python3 >=3.5)
  - currently tested with Python2 2.7 only
  - actual Python version required depends on Ansible version used and the version of Python interpreter that executes Ansible on ironic-conductor host
  - current Ansible support of Python3 on managed node is still considered experimental (and is not supported by Ansible 2.1.x at all)
- a software component that upon boot of the ramdisk will make a `lookup` API request to ironic API and make `heartbeat` requests afterwards
  - default choice for such component is IPA
- other system utilities used by deploy or clean playbooks

All or part of those (except the SSH server) can in principle be installed at runtime through additional Ansible tasks in playbooks (reducing the memory footprint and download/boot time of the deploy ramdisk), but also can be provided with deploy ramdisk to shorten the provisioning time.

Re-using IPA as `lookup` and `heartbeat` ironic API client makes it possible to have a unified deploy ramdisk for `direct`, `iscsi` and `ansible` deploy interfaces.

The `ironic-staging-drivers` project includes a set of scripts to perform a simple rebuild of TinyCore Linux based deploy ramdisk (TinyIPA), as well as `diskimage-builder` element to build a suitable

ramdisk with this utility, basing it on `ironic-agent` element and thus also containing IPA. Those will be promoted to the new `openstack/ironic-python-agent-builder` project[4].

Rebuild of CoreOS-based deploy ramdisk is currently not supported but such support can be added in the future.

Security impact

Ansible communicates with remote machines over SSH. Deploy process is secure if private SSH key is properly secured (can be accessed only by the user running ironic-conductor service).

Other end user impact

None

Scalability impact

The driver is not async and is blocking. One conductors worker thread per node being provisioned or cleaned is required. Most of the time the thread waits in blocking state for the completion of `ansible-playbook` process. Possible thread pool exhaustion must be accounted for when planning deployment that allows usage of this deploy interface and configuring thread pool size in ironic configuration file (`[conductor]workers_pool_size` option).

There are ideas on how to make the driver async / not blocking, those will be proposed in further specs/RFEs.

Performance Impact

We have conducted some tests to measure performance impact of running multiple `ansible-playbook` processes on ironic-conductor host[5].

The results show that while there is indeed a performance overhead introduced by using the `ansible` deploy interface, it is well within possibilities of quite standard hardware - we were able to provision 50 and 100 nodes concurrently via single ironic-conductor service using this deploy interface, with total provisioning time similar to `direct` deploy interface. Please see the blog post referenced above for mode details on test setup and results.

Other deployer impact

Config options

these are defined in the `[ansible]` section of ironic configuration file

verbosity
: None, 0-4. Corresponds to number of vs passed to `ansible-playbook`. Default (None) will pass vvvv when global debug is enabled in ironic, and nothing otherwise.

ansible_playbook_script
: Full path to the `ansible-playbook` script. Useful mostly for testing environments when you e.g. run Ansible from source instead of installing it. Default (None) will search in `$PATH` of the user running ironic-conductor service.

[4] https://opendev.org/openstack/ironic-python-agent-builder/
[5] https://pshchelo.github.io/ansible-deploy-perf.html

playbooks_path
Path to folder that contains all the Ansible-related files (Ansible inventory, deployment/cleaning playbooks, roles etc). Default is to use the playbooks provided with the package from where it is installed.

config_file_path
Path to Ansibles config file. When set to None will use global system default (usually `/etc/ansible/ansible.cfg`). Default is `playbooks_path`/ansible.cfg

ansible_extra_args
Extra arguments to pass to `ansible-playbook` on each invocation. Default is None.

default_username
Name of the user to use for Ansible when connecting to the ramdisk over SSH. Default is ansible. It may be overridden by per-node `ansible_username` option in nodes `driver_info` field.

default_key_file
Absolute path to the private SSH key file to use by Ansible by default when connecting to the ramdisk over SSH. If none is provided (default), Ansible will use the default SSH keys configured for the user running ironic-conductor service. Also note that private keys with password must be pre-loaded into `ssh-agent`. It may be overridden by per-node `ansible_key_file` option in nodes `driver_info` field.

default_deploy_playbook
Path (relative to $playbooks_path or absolute) to the default playbook used for deployment. Default is deploy.yaml. It may be overridden by per-node `ansible_deploy_playbook` option in nodes `driver_info` field.

default_shutdown_playbook
Path (relative to $playbooks_path or absolute) to the default playbook used for graceful in-band node shutdown. Default is shutdown.yaml. It may be overridden by per-node `ansible_shutdown_playbook` option in nodes `driver_info` field.

default_clean_playbook
Path (relative to $playbooks_path or absolute) to the default playbook used for node cleaning. Default is clean.yaml. It may be overridden by per-node `ansible_clean_playbook` option in nodes `driver_info` field.

default_clean_steps_config
Path (relative to $playbooks_path or absolute) to the default auxiliary cleaning steps file used during the node cleaning. Default is clean_steps.yaml. It may be overridden by per-node `ansible_clean_steps_config` option in nodes `driver_info` field.

extra_memory
Memory (in MiB) consumed by the Ansible-related processes in the deploy ramdisk. Affects decision if the downloaded user image will fit into RAM of the node. Default is 10.

post_deploy_get_power_state_retries
Number of times to retry getting power state to check if bare metal node has been powered off after a soft poweroff. Default is 6.

post_deploy_get_power_state_retry_interval
Amount of time (in seconds) to wait between polling power state after triggering soft poweroff. Default is 5.

The last 3 options are effectively copies of similar options in `[agent]` section of configuration file. We could use single option for (some of) those for all deploy interfaces that make use of them, especially if

we rename/move them from `[agent]` section to a section with more general name (like `[deploy]`).

Per-node fields in driver_info

These parameters can be provided with driver_info, all are optional and their default values can be set in the ironic configuration file:

ansible_username
User name to use for Ansible to access the node (default is `ansible`).

ansible_deploy_username
Deprecated in favor of `ansible_username`.

ansible_key_file
Private SSH key used to access the node. If none is provided (default), Ansible will use the default SSH keys configured for the user running ironic-conductor service. Also note that private keys with password must be pre-loaded into `ssh-agent`.

ansible_deploy_keyfile
Deprecated in favor of `ansible_key_file`.

ansible_deploy_playbook
Name of the playbook file inside the `playbooks_path` folder to use when deploying this node.

ansible_shutdown_playbook
Name of the playbook file inside the `playbooks_path` folder to use to gracefully shutdown the node in-band.

ansible_clean_playbook
Name of the playbook file inside the `playbooks_path` folder to use when cleaning the node.

ansible_clean_steps_config
Name of the YAML file inside the `playbooks_path` folder that holds description of cleaning steps used by this node, and defines playbook tags in `ansible_clean_playbook` file corresponding to each cleaning step.

Developer impact

Developers may use this deploy interface for drivers.

Implementation

Assignee(s)

Primary assignee:
Pavlo Shchelokovskyy - pas-ha (IRC), pshchelo (Launchpad)

Other contributors:
Yurii Zveryanskyy - yuriyz (IRC), yzveryanskyy (Launchpad)

Work Items

- Copy `ansible` deploy interface from `ironic-staging-drivers` project
  - most changes would happen in imports of unit test modules

- Register the `ansible` deploy interface entrypoint, add it to the list of supported deploy interfaces for `ipmi` hardware type, do not enable it by default in the configuration file.
- Copy documentation.
- Copy the `imagebuild` scripts from `ironic-staging-drivers` project to the new `ironic-python-agent-builder` project
  - Install and use scripts from this new project in DevStack plugins and gate jobs
- Amend ironics DevStack plugin to be able to set up nodes with this deploy interface.
  - Currently will require small rebuild of TinyIPA image during DevStack install.
- Copy and modify the `gate-tempest-dsvm-ironic-staging-drivers-ansible-wholedisk-ubuntu-xenia` gate job, enable it in non-voting mode on ironic project.

Dependencies

Ansible has to be installed on the host running `ironic-conductor` service.

This deploy interface was developed and tested with Ansible 2.1, and targets Ansible >= 2.1 (with some intermediate versions being excluded as incompatible). Currently the gate job testing this deploy interface passes with latest released Ansible version (2.3.2.0 as of this writing).

Also see the *Ramdisk impact* section for changes necessary to the deploy ramdisk.

Testing

- Unit tests are already in place.
- CI testing is already in place
  - as this is a vendor-agnostic deploy interface, it can be tested with virtual hardware in DevStack on upstream gates
  - the job `gate-tempest-dsvm-ironic-staging-drivers-ansible-wholedisk-ubuntu-xenial-nv` is already running in non-voting mode on all changes to `ironic-staging-drivers` project
  - it would have to be copied and modified after appropriate changes are made to ironics DevStack plugin.

Upgrades and Backwards Compatibility

None.

Documentation Impact

Documentation is already available and will have to be moved to ironic code tree as well.

References

UEFI iSCSI Boot for iLO drivers

https://bugs.launchpad.net/ironic/+bug/1526861

HPE ProLiant Servers (Gen9 and beyond) supports UEFI iSCSI Boot through its firmware. The proposed feature is to add support for this firmware based booting of an iSCSI Cinder volume in UEFI mode for Ironic iLO drivers.

Problem description

Currently, Ironic has ability to boot from Cinder volume. Moreover, this support is to boot from an iSCSI volume using bootloaders like iPXE. It doesnt provide any way to harness the feature of some servers which inherently supports booting from an iSCSI volume using their firmware capabilities. Hardware can be configured programmatically to boot from an iSCSI volume through firmware.

Proposed change

This change is based on the reference driver implementation guidelines proposed by Boot from Volume - Reference Drivers spec to support booting ironic nodes from a storage device that is hosted and/or controlled remotely. This change proposes two new methods for iLO drivers management interface; namely `set_iscsi_boot_target` and `clear_iscsi_boot_target`, which will facilitate setting and clearing iSCSI target information using iLO interfaces for UEFI iSCSI boot capable HPE Proliant servers.

The boot interface method `prepare_instance()` in `ilo` hardware type will check if the instance requested boot mode is UEFI and given volume is bootable. If so, it will set the iSCSI target in the iLO and set boot device to iSCSI target.

If the instance requested boot mode is BIOS the behavior for the two boot interfaces (`ilo-pxe` and `ilo-virtual-media`) will be as under:

- `ilo-pxe` : It will fallback to iPXE to boot the volume.
- **`ilo-virtual-media`: It will throw the following error:**
  `virtual media cannot boot volume in bios.`

The function definition for `ilo-pxe` boot interface with its pseudo-code will be as follows:

```
class IloPXEBoot(pxe.PXEBoot):

  def prepare_instance(self, task):
    """Prepares the boot of instance.

    :param task: a task from TaskManager.
    :returns: None
    :raises: IloOperationError, if some operation on iLO failed.
    """
    if deploy_utils.is_iscsi_boot(task) and boot_mode == 'uefi':
      #Call the management interface
      task.driver.management.set_iscsi_boot_target(task)
      #Set boot device to 'ISCSIBOOT'
      deploy_utils.try_set_boot_device(task, boot_devices.ISCSIBOOT)

    else:
      #Let iPXE handle this
      super(IloPXEBoot, self).prepare_instance(task)

  def clean_up_instance(self, task):
    """Cleans up the boot of instance.

    :param task: a task from TaskManager.
    :returns: None
    :raises: IloOperationError, if some operation on iLO failed.
```

(continues on next page)

(continued from previous page)

```
    """
    if deploy_utils.is_iscsi_boot(task) and boot_mode == 'uefi':
      #Call the management interface
      task.driver.management.clear_iscsi_boot_target(task)

    else:
      #Let iPXE handle this
      super(IloPXEBoot, self).clean_up_instance(task)
```

The function definition for `ilo-virtual-media` boot interface with its pseudo-code will be as follows:

```
class IloVirtualMediaBoot(base.BootInterface):

  def prepare_instance(self, task):
    """Prepares the boot of instance.

    :param task: a task from TaskManager.
    :returns: None
    :raises: IloOperationError, if some operation on iLO failed.
    """
    if deploy_utils.is_iscsi_boot(task) and boot_mode == 'uefi':
      #Call the management interface
      task.driver.management.set_iscsi_boot_target(task)
      #Set boot device to 'ISCSIBOOT'
      deploy_utils.try_set_boot_device(task, boot_devices.ISCSIBOOT)
      return

    elif deploy_utils.is_iscsi_boot(task) and boot_mode == 'bios':
      #Throw the error in bios boot mode
      msg = 'virtual media can not boot volume in bios mode.'
      raise exception.InstanceDeployFailure(msg)

    else:
      #Default code

  def clean_up_instance(self, task):
    """Cleans up the boot of instance.

    :param task: a task from TaskManager.
    :returns: None
    :raises: IloOperationError, if some operation on iLO failed.
    """
    if deploy_utils.is_iscsi_boot(task) and boot_mode == 'uefi':
      #Call the management interface
      task.driver.management.clear_iscsi_boot_target(task)
    else:
      #Fall to virtual media cleanup
```

Two new methods will be added in `ilo` drivers management interface `ilo.management.`

`IloManangement`: * set_iscsi_boot_target() - To set iSCSI target information into iLO * clear_iscsi_boot_target() - To clear iSCSI target information from iLO

New version of proliantutils library would be released that supports the above mentioned methods.

The function definition with its pseudo-code will be as follows:

```
class IloManagement(base.ManagementInterface):

  def set_iscsi_boot_target(self, task):
    """Set iscsi boot volume target info from the node.

    :param task: a task from TaskManager.
    """
    #Proliants call to set iscsi target info

  def clear_iscsi_boot_target(self, task):
    """Clear iscsi boot volume target info from the node.

    :param task: a task from TaskManager.
    """
    #Library call to clear iscsi target info
```

Alternatives

None.

Data model impact

None.

State Machine Impact

None.

REST API impact

None.

Client (CLI) impact

None.

RPC API impact

None.

Driver API impact

None.

Nova driver impact

None.

Security impact

None.

Ramdisk impact

None.

Other end user impact

None.

Scalability impact

None.

Performance Impact

None.

Other deployer impact

Deployers will be able to configure server which support UEFI iSCSI boot with this change. The workflow will be as follows:

- Operator configures the node with appropriate hardware type with boot interface and adds the capability `iscsi_boot=true` in `node.properties['capabilities']` (or it could be populated by inspection, but its not part of this spec on how it gets populated).
- Operator creates a flavor with Compute capability `iscsi_boot=true` to request bare metal booting from Cinder volume.
- Tenant creates a Cinder volume.
- Tenant requests a bare metal instance to be booted up with a Cinder volume with the above mentioned flavor.
- Node having ilo-virtual-media as boot interface with capability iscsi_boot=true should also have capability boot_mode configured to uefi only.
- Nova Ironic virt driver passes information about iSCSI volume to Ironic. For more information, refer ironic spec Add volume connection information for Ironic nodes.

Developer impact

None.

Implementation

Assignee(s)

Primary assignee:
kesper

Other contributors:
deray stendulker

Work Items

- Need to add changes in `ilo-pxe` and `ilo-virtual-media` boot interfaces.
- Need to implement `set_iscsi_boot_target` and `clear_iscsi_boot_target` in `ilo` management interface.

Dependencies

None.

Testing

This feature would be tested using HPE iLO third-party CI.

Upgrades and Backwards Compatibility

None.

Documentation Impact

iLO drivers documentation will be updated for this feature.

References

- Boot from Volume - Reference Drivers

Implement Rescue Mode

https://bugs.launchpad.net/ironic/+bug/1526449

Implement Nova rescue/unrescue in Ironic. Also implement an extension in IPA that carries out rescue-related tasks. After rescuing a node, it will be left running a rescue ramdisk, configured with the rescue_password, and listening with ssh on the specified network interfaces.

Problem description

Ironic does not currently implement the Nova rescue/unrescue interface. Therefore, end users are left with few options for troubleshooting or fixing anomalous and misconfigured nodes.

Proposed change

- Implement rescue(), and unrescue() in the Ironic virt driver (no spec reqd): https://blueprints.launchpad.net/nova/+spec/ironic-rescue-mode
- Add InstanceRescueFailure, and InstanceUnRescueFailure exceptions to Nova
- Add methods to Nova driver to poll Ironic to wait for nodes to rescue and unrescue, as appropriate.
- Store plaintext password in Ironic node instance_info for use on the rescued node.
- Add method for injecting password into OS.
- Modify Ironic state machine as described in the state machine impact section
- Add AgentRescue driver (implements base.RescueInterface). This implementation will be available only through hardware type. It would not be supported in the classic drivers as they would be deprecated shortly.
- For classic drivers rescue interface would be set to None. An exception of type UnsupportedDriverExtension would be generated when any methods of this interface are invoked for nodes managed by classic drivers.
- Add periodic task _check_rescue_timeouts to fail the rescue process if it takes longer than rescue_callback_timeout seconds for the rescue ramdisk to come online.
- Add Conductor methods: do_node_rescue, and do_node_unrescue
- Add Conductor RPC calls: do_node_rescue, and do_node_unrescue (and increment API version)
- Add conductor.rescue_callback_timeout config option
- Add rescue-related functionality to Ironic Python Agent including ability to set rescue password and kick off any needed network configuration automation
- Documentation of good practices for building rescue ramdisk in multitenant environments
- Add rescue_network configuration, which contains the UUID of the network the rescue agent should be booted onto. For security reasons, this should be separate from the networks used for provisioning and cleaning in multi-tenant environments.

An outline of the standard (non-error) rescue and unrescue processes follows:

Standard rescue process:

1. User calls Nova rescue() on a node.
2. Nova ComputeManager calls the virt drivers rescue() method, passing in rescue_password as a parameter.
3. Virt driver calls node.set_provision_state(RESCUE), with the rescue_password as a parameter.
4. Virt driver loops while waiting for provision_state to change, and updates Nova state as appropriate.
5. Ironic API receives set_provision_state call, and performs do_node_rescue RPC call (ACTIVE -> RESCUING).
6. Ironic conductor sets rescue password in instance_info and hands off call to appropriate driver.
7. Driver boots rescue ramdisk (RESCUING -> RESCUEWAIT), using the configured boot driver. As part of this process, Ironic will put the node onto the rescue_network, as configured in ironic.conf.

8. Agent ramdisk boots, performs a lookup (/v1/lookup in ironic-api), gets node info back, and begins heartbeating (/v1/heartbeat in ironic-api).
9. Upon receiving heartbeat, the conductor calls finalize_rescue (/v1/commands) with config drive and rescue password (RESCUEWAIT -> RESCUING), and removes the rescue password from the instance_info, as its no longer needed.
10. Agent sets password, configures network from information in config drive, and stops agent service.
11. The conductor flips network ports putting the node back on the tenant network, and the state is set to RESCUE.

Standard Unrescue process:

1. User calls Nova unrescue() on a node.
2. Nova calls Ironic unrescue() virt driver.
3. Virt driver calls node.set_provision_state(ACTIVE).
4. Virt driver loops while waiting for provision_state to change, and updates Nova state as appropriate.
5. Ironic API receives set_provision_state call, and performs do_node_unrescue RPC call.
6. Ironic conductor hands off call to appropriate driver.
7. Driver performs actions required to boot node normally, and sets provision state to ACTIVE.

Rescue/Unrescue with standalone Ironic:

1. Call Ironic provision state API with verb rescue, with the rescue password as an argument.
2. When finished with rescuing the instance, call Ironic provision state API with unrescue verb

Alternatives

- Continue to not support rescue and unrescue.
- Use console access to get rescue-like access into the OS, although this may not help in cases of lost password.

Data model impact

Essentially none. We will use instance_info to store, and subsequently retrieve, the rescue_password while rescuing a node.

State Machine Impact

- Add states to the Ironic state machine: RESCUING, RESCUEWAIT, RESCUE, RESCUEFAIL, UNRESCUING, UNRESCUEFAIL.
- Add transitions to the Ironic state machine:
  - ACTIVE -> RESCUING (initiate rescue)
  - RESCUING -> RESCUE (rescue succeeds)
  - RESCUING -> RESCUEWAIT (optionally, wait on external callback)
  - RESCUING -> RESCUEFAIL (rescue fails)
  - RESCUEWAIT -> RESCUING (callback succeeds)

  - RESCUEWAIT -> RESCUEFAIL (callback fails or abort issued)
  - RESCUEWAIT -> DELETING (delete instance without waiting)
  - RESCUE -> RESCUING (re-rescue node)
  - RESCUE -> DELETING (delete rescued instance)
  - RESCUE -> UNRESCUING (unrescue node)
  - UNRESCUING -> UNRESCUEFAIL (unrescue fails)
  - UNRESCUING -> ACTIVE (unrescue succeeds)
  - UNRESCUEFAIL -> RESCUING (re-rescue node after failed unrescue)
  - UNRESCUEFAIL -> UNRESCUING (re-unrescue node after failed unrescue)
  - UNRESCUEFAIL -> DELETING (delete instance that failed unrescuing)
  - RESCUEFAIL -> RESCUING (re-rescue after rescue failed)
  - RESCUEFAIL -> UNRESCUING (unrescue after failed rescue)
  - RESCUEFAIL -> DELETING (delete instance after failed rescue)
- Add state machine verbs:
  - RESCUE
  - UNRESCUE

REST API impact

Modify provision state API to support the states and transitions described in this spec. Also increment the API microversion. Nodes in states introduced by this spec (and related, future microversion) would be unable to be modified by clients using an earlier microversion.

Client (CLI) impact

ironic CLI

None, as this CLI is anticipated to be deprecated prior to this feature landing.

openstack baremetal CLI

The OSC command line will require additional code to handle the new state. New command line options `rescue` and `unrescue` would be added to support rescue and unrescue operations.

RPC API impact

Add do_node_rescue and do_node_unrescue to the Conductor RPC API.

Driver API impact

Add a new method clean_up() for RescueInterface in base.py. This method would perform any necessary clean up of the node upon RESCUEWAIT timeout/failure or finishing rescue operation. Some of the cleaning tasks are removing rescue password from the node. Ramdisk boot environment should be cleaned if ironic is managing the ramdisk boot. It would have default implementation as given below:

```
class RescueInterface(BaseInterface):
    def clean_up(self, task):
        pass
```

Nova driver impact

Implement rescue() and unrescue() in the Nova driver. Add supporting methods including _wait_for_rescue() and _wait_for_unrescue().

Ramdisk impact

An agent that wishes to support rescue should:

- Read and understand ipa-api-url kernel parameter for configuring API endpoint
- **Implement a client for ironics lookup API call**
  - The rescue_password will be in instance_info in the node object returned by Ironic on lookup. This can be placed in a linux-style /etc/shadow entry to enable a new user account.
- **Implement heartbeating to the appropriate API endpoint in Ironic**
  - After one heartbeat, the agent should then kickoff any action needed to reconfigure networking, such as re-DHCPing, as the Ironic conductor will complete all actions to finish rescue - including moving the node off a network with access to Ironic API, if relevant.
  - Once network is reconfigured, the agent process should shutdown. Rescue is complete.

IPA will have a rescue extension added, implementing the above functionality.

Security impact

The rescue_password must be sent from Nova to Ironic, and thereafter to the rescued node. If, at any step in this process, this password is intercepted or changed, an attacker can gain root access to the rescued node.

Additionally, the lookup endpoint will be required to return the rescue password as a response to the first lookup once rescue is initiated. That means a properly executed timing attack could recover the password, but since this would also cause the rescue to fail (despite the node changing states), its at worst a denial of service.

Security vulnerabilities involving the rescue ramdisk is another source of attacks. This is different from existing ramdisk issues, as once the rescue is complete, the tenant would have access to the ramdisk. This means deployers may need to ensure no secret information (such as custom cleaning steps or firmwares) are not present in the rescue ramdisk.

IPA is entirely unauthenticated. If IPA endpoints continue to be available after a node is rescued, then attackers with access to the tenant network would be able to leverage IPAs REST API to gain privileged access to the host. As such, IPA itself should be shut down, or the network should be sufficiently isolated during rescue operations.

Other end user impact

We will add rescue and unrescue commands to OSC Client.

Scalability impact

None.

Performance Impact

None.

Other deployer impact

Add conductor.rescue_callback_timeout config option.

Multi-tenant deployers will most likely need to support two ram disksone running IPA for use with normal node-provisioning tasks, and another running IPA for rescue mode (with non-rescue endpoints disabled). This is to ensure the full suite of tooling and authentication needed for secure cleaning is not given to a tenant.

Additionally, in some environments, operators may not want to use the full Ironic Python Agent inside the rescue ramdisk, due to its requirement for python or linux-centric nature. They may use statically compiled software such as onmetal-rescue-agent[0] to perform the lookup and heartbeat needed to finalize cleaning.

Developer impact

None.

Implementation

Assignee(s)

Primary assignee:
JayF

Other contributors:
Shivanand Tendulker (stendulker) Aparna (aparnavtce)

Work Items

See proposed changes.

Dependencies

- Updating the Ironic virt driver in Nova to support this.

[0] https://github.com/rackerlabs/onmetal-rescue-agent

Testing

Unit tests and Tempest tests must be added.

Upgrades and Backwards Compatibility

Clients that are unaware of rescue-related states may not function correctly with nodes that are in these states.

Documentation Impact

Write documentation.

References

Node Traits

https://bugs.launchpad.net/ironic/+bug/1722194

Add more granular Nova placement of ironic nodes by allowing operators to set a list of traits associated with each node.

Problem description

While the recent addition of Resource Class on every node has helped improve the way ironic resources are scheduled in Nova, using the new Placement API, there are many use cases that need more granular control.

An example use case is dedicating a pool of ironic nodes for a specific project. Another is allowing flavors that best fit to the available hardware, you might want smaller flavors to build on larger machines if you are out of smaller machines.

In a similar way, you might have a pool of hardware that are all the same except for the chosen RAID configuration. Longer term, when you provision the node via Nova boot it would be good to be able to automatically reconfigure the hardware with the chosen RAID configuration. For now this is considered out of scope.

Proposed change

The Placement API has the ability to tag a resource provider with a trait: http://specs.openstack.org/openstack/nova-specs/specs/pike/approved/resource-provider-traits.html

The nova ironic driver currently ensures a Resource Provider exists for every Ironic Node. It already populates the inventory with the correct quantitative resources (i.e the Resource Class).

To get the extra granularity of scheduling we depend on adding traits to the resource provider, and the implementation of the following Nova specs that ensure the traits requested in flavors are honored by the scheduling process:

- http://specs.openstack.org/openstack/nova-specs/specs/queens/approved/request-traits-in-nova.html
- http://specs.openstack.org/openstack/nova-specs/specs/queens/approved/add-trait-support-in-allocation-candidates.html

The current proposal is for Ironic to store a list of traits for each node. That list can then be synced into the appropriate resource provider in the Placement API by Novas ironic virt driver, in a similar way to how the RESOURCE_CLASS of each ironic node is reported today.

Lets talk about the use case of dedicating specific Ironic nodes for use only by a specific set of projects. The remainder of the hosts are for general use. If a user has a dedicated pool of resources, they have the ability to pick if they create an instance in their dedicated pool or in the general pool. Other users are only able to build in the general pool. One way to bisect the nodes like this is assigning traits such as *CUSTOM_IRONIC_NODE_PROJECT_B* and *CUSTOM_IRONIC_NODE_GENERAL_USE* to the appropriate ironic nodes. Then there is a public flavor to target the general pool of hosts, and a private project specific flavor that targets their dedicated pool. By taking this approach it is easy to add additional pools of nodes for other sets of projects. It is easier because you dont need to modify any of the existing nodes and flavors when you add an additional pool of nodes.

Note that inspection rules can be used to set the initial value for a nodes Resource Class. It is expected the initial set of traits for a node can be set in the same way.

Note

There is already a Nova spec proposed discussing the Nova side of how we can assign traits to the resource providers, and on Nova boot send the chosen traits back to ironic:

- http://specs.openstack.org/openstack/nova-specs/specs/queens/approved/ironic-driver-traits.html

When provisioning a node, the Ironic virt driver in Nova will now send additional flavor extra_specs to Ironic. Currently only extra_specs starting with *capabilities* are set in instance_info. After the above spec, the ironic virt driver will also include the flavor extra specs that define what traits have been requested, storing them *instance_info[traits]*.

Note

Note there is no change to how capabilities are used as part of this spec.

Ironic needs to validate that *instance_info[traits]* is consistent with the list of traits currently associated with Ironic node, i.e. we must check that *instance_info[traits]* only includes traits that are already in the list of traits set on the Ironic node. This is particularly useful in preventing strange behaviour due to races between the update of an Ironic nodes trait list and getting that list copied into the Placement API by the Ironic virt driver. This validation is probably best integrated as part of `driver.deploy.validate` or similar, to ensure it is triggered by a call to deploy a node and a call to validate a node.

Note

The ironic virt driver in Nova will sync the list of traits reported by each Ironic Node into Novas Placement API, in a similar way to how the Resource Class is done today. Should an operator talk directly to Placement to adjust the traits, this process will remove all those modifications when the next sync occurs. Operators must use the Ironic API to change the list of traits on a given node. Ironic is assumed to be the source of truth for the list of traits associated with a Resource Provider that is representing an Ironic node.

Alternatives

In the future, it is expected that a driver may need to validate some well known traits to see if they are supported. In addition it may be that some drivers automatically report some traits. Following on from that, it could be that drivers read the traits specified in instance_info to reconfigure the node. This is all out of scope here.

Looking at what is in scope, there are alternative approaches:

- Make operators (and ironic-inspector) talk directly to the Placement API to set the node traits. This would be very odd given the Nova sync of the Resource Providers using the ComputeNode uuid as the uuid for the Resource Provider. You cant set the traits until that sync has completed. Right now placement is largely an internal API with few policy controls, so that would all need to change to allow the above. It all seems very messy.
- Keep traits APIs in Ironic, but make it a pass-through proxy to Placement. This is would make Ironic hard depend on placement, which would be a strange requirement for things like Bifrost, and would complicate upgrades.

The current approach keeps Ironic from needing to depend on placement in any way, which works well.

The REST API described below allows fine grained control of setting traits, following the patterns from the API-WG and existing placement APIs. We could instead have just extended the existing ironic node PATCH interface.

The suggested CLI approach below follows the new API, but as an alternative we could have extended the existing CLI interface around the PATCH API call. It would have looked something more like:

- openstack baremetal node set trait CUSTOM_FOO <node-uuid>
- openstack baremetal node unset trait CUSTOM_FOO <node-uuid>

Data model impact

Following a similar pattern to the existing table for tags, we will need to add this new table to store the traits associated with a node:

```
CREATE TABLE node_traits (
    node_id INT(11) NOT NULL,
    trait VARCHAR(255) CHARACTER SET utf8 NOT NULL,
    PRIMARY KEY (node_id, trait),
    KEY (trait),
    FOREIGN KEY (node_id)
      REFERENCES nodes(id)
      ON DELETE CASCADE,
)
```

A new `ironic.objects.traits.NodeTraitList` object will be added to the object model. The ironic.objects.traits.NodeTraitList field in the python object model will be populated on-demand (i.e. not eager-loaded).

A trait should be defined in a way that matches the placement API definition, as a Unicode string no longer than 255 characters.

State Machine Impact

No impact.

REST API impact

The placement API defines a set of standard traits in the *os-traits* library. Any traits that are not defined in that library must start with the prefix of *CUSTOM_*. Any trait set in Ironic must follow these rules, else the ironic Nova virt driver will be unable to add the traits in Placement. For similar reasons there is a limit of 50 traits on any node, to match the limit in Placement. A request to add a badly formatted trait should get a response with the status code 400.

Note at no point does Ironic talk to the Placement API. The above validation depends only on access to the python library *os-traits*. As such, this validation poses little restriction on how traits can be used in standalone Ironic to assign arbitrary traits on particular Ironic nodes. Any non-standard traits simply need to have a prefix of `CUSTOM_` added. For more details on *os-traits* please see: https://docs.openstack.org/os-traits/latest

For convenience, it will be possible to get the full list of nodes and the traits associated with each node by extending the existing API in the following way (when requesting a high enough microversion that includes these details):

```
GET /v1/nodes/detail

{
  "nodes": [
    {
      ...
      "traits": ['CUSTOM_FOO', 'CUSTOM_BAR', 'CUSTOM_BAZ'],
      ...
    }
  ]
}
```

In a similar way to other fields, we will also support a request to get just this field (in part to make the Nova virt driver polling more efficient):

```
GET /v1/nodes/?fields=uuid,traits

{
  "nodes": [
    {
      "uuid": "uuid-1",
      "traits": ['CUSTOM_FOO', 'CUSTOM_BAR', 'CUSTOM_BAZ']
    },
    ...
  ]
}
```

The manipulation of node traits will follow the patterns established by both the placement API and API WG tags spec:

- https://developer.openstack.org/api-ref/placement/#resource-provider-traits

- http://specs.openstack.org/openstack/api-wg/guidelines/tags.html

To start with there will be a new traits resource that follows the above patterns.

Example request for all node traits:

```
GET /nodes/{node_ident}/traits
```

Response:

```
{
    "traits": ['CUSTOM_FOO', 'CUSTOM_BAR', 'CUSTOM_BAZ']
}
```

Example request to set all node traits to given list:

```
PUT /nodes/{node_ident}/traits
{
    "traits": ['CUSTOM_FOO', 'CUSTOM_BAR', 'CUSTOM_BAZ']
}
```

Response:

```
{
    "traits": ['CUSTOM_FOO', 'CUSTOM_BAR', 'CUSTOM_BAZ']
}
```

The response on success is status code 200. On failure to validate (using the os-traits library) we return the status code 400 (Bad Request), matching the HTTP Guidelines from the API-WG.

Note that unlike with Resource Class, we are allowing the trait to be updated at any time. This is mostly because placement allows such updates and because although the Resource Class and Ironic node are used in the allocations in placement, traits are not used in allocations.

In a similar way the following API removes all the traits:

```
DELETE /nodes/{node_ident}/traits
```

The response on success is status code 204, with an empty body.

To add or remove an individual trait use:

```
PUT /nodes/{node_ident}/traits/CUSTOM_FOO
<no body>

DELETE /nodes/{node_ident}/traits/CUSTOM_FOO
```

Filtering the node list by traits should work as expected:

```
GET /nodes?traits=CUSTOM_RED,CUSTOM_BLUE
GET /nodes?not-traits=CUSTOM_RED,CUSTOM_BLUE&traits=CUSTOM_FOO
GET /nodes?traits-any=CUSTOM_RED,CUSTOM_BLUE
GET /nodes?not-traits-any=CUSTOM_RED,CUSTOM_BLUE
```

As mentioned above, the final change that is made is to ensure `instance_info/traits` is a subset of the traits set on the Ironic node. This should be part of the existing `driver.deploy.validate()` call

(or similar) such that the traits will be checked both before a deploy starts and on an explicit node validate call.

Client (CLI) impact

ironic CLI

No changes, it is deprecated.

openstack baremetal CLI

You can list the traits on a node:

- openstack baremetal node list fields uuid name traits
- openstack baremetal node show <node-ident> fields uuid name traits
- openstack baremetal node trait list <node-ident>

You can update the list of traits on a node:

- openstack baremetal node add trait <node-ident> CUSTOM_FOO CUSTOM_BAR
- openstack baremetal node remove trait <node-ident> CUSTOM_FOO CUSTOM_BAR
- openstack baremetal node remove trait all <node-ident>

This is roughly copying the command syntax of consistency groups: https://docs.openstack.org/python-openstackclient/latest/cli/command-objects/consistency-group.html#consistency-group-add-volume

It is common to use set and unset for key value pairs, but add and remove seems a better fit for in-place modifications of a list. It stops any ambiguity of set meaning either an addition of a list of traits or replacing the whole list of traits. Another alternative is to add trait operations into the existing `openstack baremetal node set` operation, but we are instead following the structure of the API.

You can query the list of nodes using traits:

- openstack baremetal node list trait CUSTOM_RED not-trait CUSTOM_BLUE
- openstack baremetal node list trait-any CUSTOM_RED CUSTOM_BLUE
- openstack baremetal node list not-trait-any CUSTOM_RED CUSTOM_BLUE

RPC API impact

No impact.

Driver API impact

No impact.

Nova driver impact

Need to ensure the correct flavor extra specs are passed back when starting a node.

Ramdisk impact

None

Security impact

There will be a hard coded limit of 50 traits for any Node to prevent misuse of the API. This prevents denial of service attack where the database is filled up by a rogue user setting lots of traits. Really the limit is in place to match the limit applied in the placement API.

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

None

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
John Garbutt (johnthetubaguy)

Other contributors:
Dmitry Tantsur (dtantsur) Mark Goddard

Work Items

- Add table to store traits for nodes
- Add object to expose the table
- Add new traits API
- Add openstack cli support for the new API
- Follow up with Nova driver work

Dependencies

The following nova spec depends on this spec:

- http://specs.openstack.org/openstack/nova-specs/specs/queens/approved/ironic-driver-traits.html

Testing

Nova functional tests are planning on covering the scheduling aspects of the integration. As part of this spec we will focus on ensuring the API works correctly to persist the traits for given nodes, and query resources using traits.

Upgrades and Backwards Compatibility

Longer term, capabilities and other APIs may be phased out. We are not considering that as part of this spec. There is much more work needed before we have feature parity between the old and new scheduling mechanisms.

Documentation Impact

Need to update the API-REF and the admin doc to cover how to use the new API.

References

- http://specs.openstack.org/openstack/nova-specs/specs/pike/approved/resource-provider-traits.html
- http://specs.openstack.org/openstack/nova-specs/specs/queens/approved/ironic-driver-traits.html
- http://specs.openstack.org/openstack/nova-specs/specs/queens/approved/request-traits-in-nova.html
- http://specs.openstack.org/openstack/nova-specs/specs/queens/approved/add-trait-support-in-allocation-candidates.html

Lenovo XClarity Driver

https://bugs.launchpad.net/ironic/+bug/1702508

This specification proposes to add new interfaces that provide Ironic support to Lenovo XClarity managed servers. Servers that are supported by XClarity today are provided in the references at the end of this specification.

Problem description

Lenovo servers use an IMM GUI which brings forth a unique way of managing the nodes. The nodes need to be created, managed and operated by the XClarity Administrator GUI tool and are identified by a unique host id.

In addition to managing the nodes using IPMI protocol, this specification proposes to add hardware types and interfaces to manage Lenovo servers using XClarity Administrator RESTful API. The REST API documentation for XClarity is provided in the references at the end of the document. Benefits of using XClarity over plain IPMI are: better user management, hardware monitoring and hardware management.

Proposed change

New power and management interfaces will be added as part of this change.

The interfaces use RESTful API to communicate with XClarity Administrator. The interfaces used are:

- XClarity.XClarityPower for Power operations
- XClarity.XClarityManagement for Management operations

The XClarity Administrator RESTful API provides various operations, like controlling the power of nodes, retrieving firmware version and Feature on Demand(FoD) enablement information, etc.

The multiple interfaces embed the client side code for communicating with XClarity API via HTTP/HTTPS protocol, as a simple wrapper class. For all the interfaces, an xclarity-client will to be imported. The xclarity-client is available at https://pypi.python.org/simple/xclarity-client/.

- Power:

  This feature allows the user to turn the node on/off or reboot by using the power interface which will in turn call XClaritys REST API.

- Management:

  This feature allows the user to get and set the primary boot device of the Lenovo servers, and to get the supported boot devices.

Alternatives

Use of the generic IPMI interfaces and pre-existing deploy interfaces is an alternative especially in mixed configurations.

Data model impact

None

RPC API impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

Driver API impact

None

Nova driver impact

None

Security impact

Use of HTTPS for this interface, and verifying certificates is more secure than IPMI-based interfaces.

Lenovo XClarity Administrator uses certificates to establish secure, trusted communications between XClarity and its managed devices. By default, the managed devices use XClarity-generated certificates. The user can choose to let Lenovo XClarity Administrator manage certificates, or customize and replace the server certificates. XClarity provides options for customizing certificates depending on the users requirements.

Other end user impact

None

Scalability impact

There are some locking considerations when we are dealing with scale. All locking is within a primary job/task framework which are profile activation, firmware update, and OS deploy. A client will initiate a job on an XClarity API endpoint, at which time it is locked from other jobs or client actions until the job completes.

There is an additional mechanism used by Firmware Updates to prevent inventory from refreshing from events, so commands are not sent to the IMM during the update. A second level of locking is when a client attempts power actions, and during the power request the endpoint is locked. This is generally very quick though.

Performance Impact

Lenovo XClarity Administrator supports the management of up to 20 chassis with compute nodes and a similar number of rack servers. An operator with a large number of systems being managed by XClarity should expect reduced system performance. Performance considerations have been provided in references at the end of this specification.

Ramdisk impact

None

Other deployer impact

The following driver_info fields are required while enrolling node into Ironic:

- xclarity_address: XClarity Administrator IP-Address
- xclarity_username: XClarity Administrator username
- xclarity_password: XClarity Administrator password

- xclarity_hostid: The host ID that is allocated by XClarity Administrator for each managed host.
- xclarity_port(optional): The port used for establishing xClarity Administrator connection. Default is 443.

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
Hu Bian (hubian1@lenovo.com)

Other contributors:
Jia Pei (jiapei2@lenovo.com) Haijun Mao (maohj2@lenovo.com) Finix Lei (leilei4@lenovo.com) Rushil Chugh (crushil)

Work Items

- Add new XClarity hardware type, and adding new interfaces for Power and Management.
- Writing appropriate unit tests to provide test coverage for XClarity driver.
- Writing configuration documents.
- Third party CI-setup.

Dependencies

None

Testing

- Unit tests will be implemented for new XClarity driver.
- Third party Continuous integration (CI) support will be added for Lenovo servers.

Upgrades and Backwards Compatibility

None

Documentation Impact

- Updating Ironic documentation section *Enabling Drivers* with XClarity related instructions.

References

- XClarity Restful API introduction
- XClarity Supported servers
- XClarity Performance considerations <http://flexsystem.lenovofiles.com/help/index.jsp?topic=%2Fcom.lenovo.lxca.doc%2Fplan_performconsiderations.html>_

5.8.2 10.0

ironic-python-agent API Versioning and Negotiation

https://bugs.launchpad.net/ironic/+bug/1602265

It was decided at the ironic newton midcycle that we need some form of API version negotiation between ironic and IPA. This is required to allow ironic to recognise that it is talking to an older ramdisk, and that newer features will not be available.

Problem description

During ironic upgrade its possible that you will end up with node which uses an version of the ironic-python-agent (IPA) older than the version of ironic. In this case newer features that have been added to ironic and IPA will be expected to exist by ironic however because the ramdisk used on a node has not been upgraded yet, those features do not exist in that version of IPA. This leads to failures because ironic tries to use these features during deployment/cleaning and resulting in unexpected responses from IPA.

Proposed change

The proposed change is that when IPA does its heartbeat to ironic it will include an IPA version number that will be cached by ironic in the nodes `driver_internal_info` in the same way as the Agent URL is stored. ironic will then use this version information to gracefully degrade the feature set that it uses from the ironic-python-agent. For example:

```
def do_something_using_IPA(self)
  # Make sure to refresh the node object in case agent_version has changed
  # via a heartbeat since we first fetched the node.
  node = self.node.refresh()
  if node.driver_internal_info.get('agent_version', '0.0.0') >= '1.2':
    do_new_additional_thing_only_supported_in_1.2()
  do_thing_using_IPA()
```

Alternatives

A couple of alternatives exist:

- Require that operators update every nodes ramdisk to the newer version of the ramdisk with the new features before upgrading their ironic installation.
- Handle the cases where IPA doesnt support a particular feature by catching a set of expected errors on a case by case basis and gracefully fall back to another method. However in cases where this occurs it will result in more complex error handling in Ironic and extra API calls to IPA.

Data model impact

New `agent_version` field will be stored in `node.driver_internal_info` on agent heartbeat.

State Machine Impact

In the existing code the cleaning step checks the IPA hardware manager version and if it changes for any reason during the cleaning process, then cleaning is aborted and will need to be restarted. The agent version being added by this spec should be treated in the same way to ensure the most stable environment for cleaning.

REST API impact

A new field `agent_version` will we included in the body of the heartbeat API request alongside the existing `agent_url` field.

Client (CLI) impact

None

ironic CLI

None

openstack baremetal CLI

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

IPA will need updating to include its current version in the `agent_version` field when it makes a heartbeat request.

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

This change will allow for deployers upgrading their ironic installs to upgrade their nodes ironic-python-agent ramdisks asynchronously from the rest of ironic.

Developer impact

Developers of ironic drivers that interact with the ironic-python-agent will need to ensure their code takes into account the agent version that they might be talking to. Making sure that if talking to an IPA that does not support a feature it disables that feature in ironic.

Implementation

Assignee(s)

Primary assignee:
sambetts

Work Items

- Add code to IPA to send its version to ironic in the heartbeat request
- Add code to ironic to accept and store the `agent_version` when it receives a heartbeat
- Add developer documentation on the correct way to add support for an IPA feature in ironic, such that it will gracefully degrade if it is not available.

Dependencies

None

Testing

- ironic grenade tests already test ironic master with the last named release IPA
- Normal ironic/IPA tests will test ironic master + IPA master builds
- Need to add a grenade test to IPA to test last ironic named release + master IPA

Upgrades and Backwards Compatibility

- This spec adds the ability to better support Ironic version N+1 with IPA version N or older as Ironic will gracefully degrade which features it will request if they arent available
- Ironic version N works with IPA version N+1 and should continue to work

Documentation Impact

- Need to add the developer documentation mention in the work items section
- Need to document which versions of IPA are supported with which versions of ironic.

References

- https://etherpad.openstack.org/p/ironic-newton-midcycle

5.8.3 9.2

Change CLI default API version to latest known ironic API version

https://bugs.launchpad.net/python-ironicclient/+bug/1671145

When using ironic CLI or the OpenStack CLI (OSC), ironic API version 1.9 is used by default. This spec proposes raising the default ironic API version used in both CLIs to the latest compatible API version.

Problem description

Currently, ironic CLI and the baremetal OSC plugin default to using an API version that dates to the Liberty release.[1]

This means that any new user using the CLI who doesnt specify the version can only use features that are almost 2 years old as of the time of this writing. This limits discoverability and usability of new features.

Also, if we ever bump up the minimum supported API version in the API, we will have to deal with raising the minimum CLI version anyway.

Proposed change

- For the Pike cycle, a warning message will be printed when a user runs either CLI without specifying the version, indicating that the CLI will soon default to using the latest compatible API version. The proposed wording for the OSC plugin is as follows:

  > You are using the default API version of the OpenStack CLI baremetal (ironic) plugin. This is currently API version 1.9. In the future, the default will be the latest API version understood by both API and CLI. You can preserve the current behavior by passing the os-baremetal-api-version argument with the desired version or using the OS_BAREMETAL_API_VERSION environment variable.

  A similar wording will be used for the `ironic` tool:

  > You are using the default API version of the `ironic` CLI tool. This is currently API version 1.9. In the future, the default will be the latest API version understood by both API and CLI. You can preserve the current behavior by passing the ironic-api-version argument with the desired version or using the IRONIC_API_VERSION environment variable.

  If the user wishes to continue using API version 1.9, they should specify so on the command line.

- During the Queens cycle, the default API version used by the CLI will change to the latest version of the API compatible with the CLI. If only the major version is specified (for example, `--os-baremetal-api-version=1`), the latest compatible version of that major version will be used (e.g. 1.32 if thats the latest 1.XX version).

  The latest compatible version will be determined via version negotiation between the CLI and ironic. The CLI will first make a request to the root endpoint of the ironic API, which returns the version of the ironic API.[2] If the ironic API version is lower than the maximum version the client is

[1] https://docs.openstack.org/developer/ironic/dev/webapi-version-history.html
[2] https://etherpad.openstack.org/p/ironic-pike-ptg-operations

compatible with, the CLI will use the version running on the API service to ensure compatibility. Otherwise, the maximum ironic API version that can be handled by the client will be used.

> **Note**
>
> We may deprecate the `ironic` tool in the Queens cycle. If this happens, we may decide to skip applying this change to this tool, and proceed with the deprecation instead. This is subject to a separate spec.

Changes to the client library are out of scope for this spec.

This change was discussed in detail at the Pike PTG.[3]

Alternatives

- We could periodically bump the default API version used by the CLI. This has the disadvantage of being unpredictable from a user standpoint and doesnt solve the discoverability issue for the latest features. Its also not ideal for maintainability of the CLI codebase.
- We could keep the default as 1.9 and always log a warning, without ever changing the default to the latest version. This doesnt solve the ease of use or discoverability issues, and it permanently adds an extra annoyance to users who dont wish to specify a version every time the CLI is invoked.
- We could just pass `latest` to the API when no version is provided. However, this approach would lead to potentially broken behavior if we ever land a breaking change to our API.

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

ironic CLI

A warning will be logged when ironic CLI is invoked without specifying `--ironic-api-version`, indicating that the default will soon be the latest compatible API version.

After the deprecation period, the default will be changed accordingly, assuming that ironic CLI itself is not deprecated beforehand.

[3] https://developer.openstack.org/api-ref/baremetal/#list-api-versions

openstack baremetal CLI

A warning will be logged when the OSC baremetal plugin is invoked without specifying `--os-baremetal-api-version`, indicating that the default will soon be the latest compatible API version.

After the deprecation period, the default will be changed accordingly.

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

None

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

None

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
dtantsur

Other contributors:
mariojv

Work Items

- Add the warning message
- After the standard deprecation period (release boundary or 3 months, whichever is longer), change the default API version used by the CLI.

Dependencies

None

Testing

Appropriate unit and functional testing will be added.

Upgrades and Backwards Compatibility

If a user has scripts or tooling that use the CLI without specifying the version, those will need to be updated to specify the version.

Documentation Impact

None. Appropriate release notes will be added.

References

5.9 Pike

5.9.1 9.1

Add more capabilities to ironic inspection

https://bugs.launchpad.net/ironic/+bug/1599425

This spec adds a few more capabilities to ironic drivers. The capabilities can be implemented in-band or out-of-band as per driver maintainers technical decision.

Problem description

The operator may be interested to schedule and select the hardware based on many other hardware properties like cpu_vt, trusted_boot, etc.

Proposed change

Following properties will be discovered:

```
capability name: trusted_boot
possible values: true|false
explanation    : The hardware supports trusted boot or not.
```

(continues on next page)

(continued from previous page)

```
capability name: iscsi_boot
possible values: true|false
explanation    : The hardware supports iscsi boot or not.

capability name : boot_mode_uefi
possible values : true|false
explanation     : The hardware supports uefi boot or not.
                  It is not the current boot mode. This
                  may not be discovered through inband
                  inspection.

capability name : boot_mode_bios
possible values : true|false
explanation     : The hardware supports bios boot or not.
                  It is not the current boot mode.

capability name : sriov_enabled
possible values : true|false

capability_name : has_ssd
possible values : true|false

capability_name : has_rotational
possible_values : true|false

capability name : rotational_drive_<rpm_value>_rpm
possible values : true|false
explanation     : The capabilities will turn out to be
                  rotational_drive_4800_rpm,
                  rotational_drive_5400_rpm,
                  rotational_drive_7200_rpm,
                  rotational_drive_10000_rpm, and
                  rotational_drive_15000_rpm. These
                  looks to be the only and standard values
                  for rotational drive rpms.

capability name : logical_raid_level_<num>
possible values : true|false
explanation     : The capabilities ``logical_raid_level_<num>``
                  will have "num" as the dynamic number and
                  will have names as ``logical_raid_level_1``,
                  ``logical_raid_level_2``, and so on. There can
                  be multiple RAIDs configured on a hardware. This
                  gives flexibility to the operator to choose a
                  hardware which has the desired raid configured.
                  So if RAID level 1 is configured, the
                  variable becomes ``logical_raid_level_1`` set
                  to ``true``. if RAID level 5 is configured,
                  the variable becomes ``logical_raid_level_5``
```

(continues on next page)

(continued from previous page)

```
                  set to ``true``. These capabilities would be
                  used only for scheduling, and ironic is
                  not supposed to create RAID level as per these
                  capabilities.
                  These capabilities should be added via RAID
                  interfaces also. These are added via inspection
                  as there can be baremetals added to ironic which
                  have RAID pre-configured.

capability name : cpu_vt
possible values : true|false

capability name : hardware_supports_raid
possible values : true|false

capability name : boot_mode
possible values : bios, uefi
explanation     : This represents the deploy boot mode of
                  the system.

capability name : has_nvme_ssd
possible values : true|false

capability name : persistent_memory
possible values : true|false

capability name : nvdimm_n
possible values : true|false

capability name : logical_nvdimm_n
possible values : true|false
```

The other drivers/vendors may require the below list of capabilities. These are already implemented by iLO drivers:

```
capability name : <driver>_firmware_version
possible values : varies from hardware to hardware
explanation     : Here driver means ilo or irmc or any other
                  vendor. Hence the capability becomes
                  ilo_firmware_version or irmc_firmware_version.

capability name : server_model
possible values : varies from hardware to hardware.

capability name : secure_boot
possible values : true|false

capability name : pci_gpu_devices
possible values : count of total GPU devices.
```

(continues on next page)

(continued from previous page)

```
capability name : nic_capacity
possible values : Maximum NIC capacity value with unit.

capability name : rom_firmware_version
possible values : vary from hardware to hardware.
```

Few which are already implemented by ironic-inspector:

```
capability name : cpu_aes
possible values : true|false

capability name : cpu_txt

capability name : cpu_hugepages

capability name : cpu_hugepages_1g
```

These may not be part of capabilities specifically, but required to be inspected. The ironic-inspector already inspects these properties:

```
property name   : switch_id
explanation     : Identifies a switch and can be a MAC address
                  or an OpenFlow-based ``datapath_id``

property_name   : port_id
explanation     : Port ID on the switch, for example, Gig0/1

property_name   : switch_info
explanation     : Used to distinguish different switch models
                  or other vendor-specific identifier.
```

`has_ssd` and `has_rotational` are two different properties as the hardware can have both kind of drives attached.

The capabilities `boot_mode_*` are added as a hardware could be supporting both bios and uefi and the current capability `boot_mode` can accept only one value. The drivers would need to adjust the deploy behaviour when the new capabilities `boot_mode_bios` or `boot_mode_uefi` are given in nova flavor. The changes required in drivers for `boot_mode_*` capabilities is out of scope of this spec.

Alternatives

Operator may need to manually configure the node with above properties for nova scheduler to be able to select the desired node for deploy.

Data model impact

None.

State Machine Impact

None.

REST API impact

None.

Client (CLI) impact

None.

RPC API impact

None.

Driver API impact

None.

Nova driver impact

None.

Ramdisk impact

None.

Security impact

None.

Other end user impact

None.

Scalability impact

None.

Performance Impact

None.

Other deployer impact

None.

Developer impact

None.

Implementation

Assignee(s)

Primary assignee:
Nisha Agarwal <agarwalnisha1980>

Work Items

- To add the above capabilities to the inspection.

Dependencies

None.

Testing

- Test the drivers to return above properties after inspection is done.

Upgrades and Backwards Compatibility

None.

Documentation Impact

Following will be documented:

- The new properties which would be added as part of this spec.
- The nova flavor samples how these properties can be used in creation of required nova flavors.

References

None.

Support Rolling Upgrade

https://bugs.launchpad.net/ironic/+bug/1526283

This proposes support for rolling upgrades of ironic, which will allow operators to roll out new code to the ironic services without having to restart all services on the new code simultaneously. There could be minimal downtime when upgrading ironic services. During the upgrade of ironic, instances will not be impacted; they should continue to run and have network access. There might be slightly longer delays for ironic operations to be performed on baremetal nodes though.

This support for rolling upgrade will satisfy the criteria mentioned in the OpenStack governance[1].

[1] https://github.com/openstack/governance/blob/master/reference/tags/assert_supports-rolling-upgrade.rst

Problem description

People operating an OpenStack cloud face the same problem: how to upgrade the cloud to the latest version so as to enjoy the new features and changes.

The upgrade of a cloud is typically done one component at a time. For example, these components of OpenStack could be upgraded in this order:

1. upgrade keystone
2. upgrade glance
3. upgrade ironic
4. upgrade nova
5. upgrade neutron
6. upgrade cinder

Ironic already supports cold upgrades[14], where the ironic services have to be down during the upgrade. For time-consuming upgrades, it may be unacceptable for the services to be unavailable for a long period of time. The steps[13] to do a cold upgrade could be:

1. stop all ironic-api and ironic-conductor services
2. uninstall old code
3. install new code
4. update configurations
5. DB sync (most time-consuming task)
6. start ironic-api and ironic-conductor services

A rolling upgrade would provide a better experience for the users and operators of the cloud. In the context of ironics rolling upgrade, it means it wouldnt be necessary to upgrade all the ironic-api and ironic-conductor services simultaneously as in a cold upgrade. A rolling upgrade would allow individual ironic-api and ironic-conductor services to be upgraded one at a time, with the rest of the services still available. This upgrade would have minimal downtime.

Although wed like the rolling upgrade solution presented here to be the same as that used by novas upgrade process[2], there are some differences between nova and ironic that prevent us from using the same solution. The differences are mentioned below, in other sections of this specification.

There have been several discussions about rolling upgrades ([3],[9],[10]).

Proposed change

For a rolling upgrade to have minimal downtime, there should be at least two ironic-api services and two ironic-conductor services running.

Since ironic doesnt support database downgrades, rollbacks will not be supported.

[14] https://github.com/openstack/governance/blob/master/reference/tags/assert_supports-upgrade.rst
[13] http://docs.openstack.org/developer/ironic/deploy/upgrade-guide.html
[2] http://docs.openstack.org/developer/nova/upgrade.html
[3] https://etherpad.openstack.org/p/ironic-mitaka-midcycle
[9] http://lists.openstack.org/pipermail/openstack-dev/2016-April/092773.html
[10] https://etherpad.openstack.org/p/ironic-newton-summit-live-upgrades

To support rolling upgrade for ironic, the following are needed:

- code changes
- a contribution guide for developers and reviewers
- a multi-node grenade CI to prevent breaking the mechanism of rolling upgrade
- a rolling upgrade operation guide for operators.

The following sub-sections describe:

- support for rolling upgrades between ironic releases
- the proposed rolling upgrade process
- the changes that are needed

Rolling upgrades between releases

Ironic follows the release-cycle-with-intermediary release model[6]. The releases are semantic-versioned[7], in the form <major>.<minor>.<patch>. We refer to a named release of ironic as the release associated with a development cycle like Mitaka.

In addition, ironic follows the standard deprecation policy[8], which says that the deprecation period must be at least three months and a cycle boundary. This means that there will never be anything that is both deprecated *and* removed between two named releases.

Rolling upgrades will be supported between:

- Named release N to N+1. (N would start with Newton if this feature is merged in Ocata.)
- Any named release to its latest revision, containing backported bug fixes. Because those bug fixes can contain improvements to the upgrade process, the operator should patch the system before upgrading between named releases.
- Most recent named release N (and semver releases newer than N) to master. As with the above bullet point, there may be a bug or a feature introduced on a master branch, that we want to remove before publishing a named release. Deprecation policy allows to do this in a 3 month time frame[8]. If the feature was included and removed in intermediate releases, there should be a release note added, with instructions on how to do a rolling upgrade to master from an affected release or release span. This would typically instruct the operator to upgrade to a particular intermediate release, before upgrading to master.

Rolling upgrade process

The rolling upgrade process to upgrade ironic from version `FromVer` to the next version `ToVer` is as follows:

1. Upgrade Ironic Python Agent image before upgrading ironic.
2. Upgrade DB schema to `ToVer` via **ironic-dbsync upgrade**. Ironic already has the code in place to do this. However, a new DB migration policy (described below in *New DB model change policy*) needs to be documented.

[6] https://releases.openstack.org/reference/release_models.html
[7] http://semver.org/
[8] http://governance.openstack.org/reference/tags/assert_follows-standard-deprecation.html

3. Pin RPC and IronicObject versions to the same `FromVer` for both ironic-api and ironic-conductor services, via the new configuration option described below in *RPC and object version pinning*.
4. Upgrade code and restart ironic-conductor services, one at a time.
5. Upgrade code and restart ironic-api services, one at a time.
6. Unpin RPC and object versions so that the services can now use the latest versions in `ToVer`. This is done via updating the new configuration option described below in *RPC and object version pinning* and then restarting the services. ironic-conductor services should be restarted first, followed by the ironic-api services. This is to ensure that when new functionality is exposed on the unpinned API service (via API micro version), it is available on the backend.
7. Run a new command **ironic-dbsync online_data_migration** to ensure that all DB records are upgraded to the new data version. This new command is discussed in a separate RFE[12] (and is a dependency for this work).
8. Upgrade ironic client libraries (e.g. python-ironicclient) and other services which use the newly introduced API features and depend on the new version.

The above process will cause the ironic services to be running the `FromVer` and `ToVer` releases in this order (where step refers to the steps above):

step	ironic-api	ironic-conductor
0	all FromVer	all FromVer
4.1	all FromVer	some FromVer, some ToVer-pinned
4.2	all FromVer	all ToVer-pinned
5.1	some FromVer, some ToVer-pinned	all ToVer-pinned
5.2	all ToVer-pinned	all ToVer-pinned
6.1	all ToVer-pinned	some ToVer-pinned, some ToVer
6.2	all ToVer-pinned	all ToVer
6.3	some ToVer-pinned, some ToVer	all ToVer
6.4	all ToVer	all ToVer

New DB model change policy

This is not a code change but it impacts the SQLAlchemy DB model and needs to be documented well for developers as well as reviewers. This new DB model change policy is as follows:

- Adding new items to the DB model is supported.
- The dropping of columns/tables and corresponding objects fields is subject to ironic deprecation policy[Page 438, 8]. But its alembic script has to wait one more deprecation period, otherwise an unknown column exception will be thrown when `FromVer` services access the DB. This is because **ironic-dbsync upgrade** upgrades the DB schema but `FromVer` services still contain the dropped field in their SQLAlchemy DB model.
- alter_column like rename or resize is not supported anymore. This has to be split into multiple operations, like add column, then remove column. Some changes may have to be split into multiple releases to maintain compatibility with an old SQLAlchemy model.
- some implementations of ALTER TABLE like adding foreign keys in PostgreSQL may impose table locks and cause downtime. If the change cannot be avoided and the impact is significant (the

[12] http://bugs.launchpad.net/ironic/+bug/1585141

table can be frequently accessed and/or store a large dataset), these cases must be mentioned in the release notes.

RPC and object version pinning

For the ironic (ironic-api and ironic-conductor) services to be running old and new releases at the same time during a rolling upgrade, the services need to be able to handle different RPC versions and object versions.

[4] has a good description of why we need RPC versioning, and describes how nova deals with it. This proposes taking a similar approach in ironic.

For object versioning, ironic uses oslo.versionedobjects.[5] describes novas approach to the problem. Unfortunately, ironics solution is different, since ironic has a more complex situation. In nova, all database access (reads and writes) is done via the nova-conductor service. This makes it possible for the nova-conductor service to be the only service to handle conversions between different object versions. (See[5] for more details.) Given an object that it doesnt understand, a (non nova-conductor) service will issue an RPC request to the nova-conductor service to get the object converted to its desired target version. Furthermore, for a nova rolling upgrade, all the non-nova-compute services are shut down, and then restarted with the new releases; nova-conductor being the first service to be restarted (Page 437, 2). Thus, the nova-conductor services are always running the same release and dont have to deal with differing object versions amongst themselves. Once they are running the new release, they can handle requests from other services running old or new releases.

Contrast that to ironic, where both the ironic-api and ironic-conductor services access the database for reading and writing. Both these services need to be aware of different object versions. For example, ironic-api can create objects such as Chassis, Ports, and Portgroups, saving them directly to the database without going through the conductor. We cannot take down the ironic-conductor in a similar way as the nova-conductor service, because ironic-conductor does a whole lot more than just interacting with the database, and at least one ironic-conductor needs to be running during a rolling upgrade.

A new configuration option will be added. It will be used to pin the RPC and IronicObject (e.g., Node, Conductor, Chassis, Port, and Portgroup) versions for all the ironic services. With this configuration option, a service will be able to properly handle the communication between different versions of services.

The new configuration option is: `[DEFAULT]/pin_release_version`. The default value of empty indicates that ironic-api and ironic-conductor will use the latest versions of RPC and IronicObjects. Its possible values are releases, named (e.g. `ocata`) or sem-versioned (e.g. `7.0`).

Internally, ironic will maintain a mapping that indicates the RPC and IronicObject versions associated with each release. This mapping will be maintained manually. (It is possible, but outside the scope of this specification, to add an automated process for the mapping.) Here is an example:

- objects_mapping:

```
{'mitaka': {'Node': '1.14', 'Conductor': '1.1',
            'Chassis': '1.3', 'Port': '1.5', 'Portgroup': '1.0'},
 '5.23': {'Node': '1.15', 'Conductor': '1.1',
          'Chassis': '1.3', 'Port': '1.5', 'Portgroup': '1.0'}}
```

- rpc_mapping:

[4] http://superuser.openstack.org/articles/upgrades-in-openstack-nova-remote-procedure-call-apis
[5] http://superuser.openstack.org/articles/upgrades-in-openstack-nova-objects/

```
{'mitaka': '1.33', '5.23': '1.33'}
```

During a rolling upgrade, the services using the new release should set this value to be the name (or version) of the old release. This will indicate to the services running the new release, which RPC and object versions that they should be compatible with, in order to communicate with the services using the old release.

Handling RPC versions

`ConductorAPI.__init__()` already sets a `version_cap` variable to the latest RPC API version and passes it to the `RPCClient` as an initialization parameter. This `version_cap` is used to determine the maximum requested message version that the `RPCClient` can send.

In order to make a compatible RPC call for a previous release, the code will be changed so that the `version_cap` is set to a pinned version (corresponding to the previous release) rather than the latest `RPC_API_VERSION`. Then each RPC call will customize the request according to this `version_cap`.

Handling IronicObject versions

Internally, ironic services (ironic-api and ironic-conductor) will deal with IronicObjects in their latest versions. Only at these boundaries, when the IronicObject enters or leaves the service, will we need to deal with object versioning:

- *getting objects from the database*: convert to latest version
- *saving objects to the database*: if pinned, save in pinned version; else save in latest version
- *serializing objects (to send over RPC)*: if pinned, send pinned version; else send latest version
- *deserializing objects (receiving objects from RPC)*: convert to latest version

The ironic-api service also has to handle API requests/responses based on whether or how a feature is supported by the API version and object versions. For example, when the ironic-api service is pinned, it can only allow actions that are available to the objects pinned version, and cannot allow actions that are only available for the latest version of that object.

To support this:

- add a new column named `version` to all the database tables (SQLAlchemy models) of the IronicObjects. The value is the version of the object that is saved in the database.

  This version column will be null at first and will be filled with the appropriate versions by a data migration script. If there is a change in Ocata that requires migration of data, we will check for null in the new version column.

  No project uses the version column mechanism for this purpose, but it is more complicated without it. For example, Cinder has a migration policy which spans 4 releases in which data is duplicated for some time. Keystone uses triggers to maintain duplicated data in one release cycle. In addition, the version column may prove useful for zero-downtime upgrades (in the future).

- add a new method `IronicObject.get_target_version(self)`. This will return the target version. If pinned, the pinned version is returned. Otherwise, the latest version is returned.

- add a new method `IronicObject.convert_to_version(self, target_version)`. This method will convert the object into the target version. The target version may be a newer or older version that the existing version of the object. The bulk of the work will be done in the new helper

method `IronicObject._convert_to_version(self, target_version)`. Subclasses that have new versions should redefine this to perform the actual conversions.

- add a new method `IronicObject.do_version_changes_for_db(self)`. This is described below in *Saving objects to the database (API/conductor > DB)*.
- add a new method `IronicObjectSerializer._process_object(self, context, objprim)`. This is described below in *Receiving objects via RPC (API/conductor <- RPC)*.

In the following,

- The old release is `FromVer`; it uses version 1.14 of a Node object.
- The new release is `ToVer`; it uses version 1.15 of a Node object this has a deprecated `extra` field and a new `meta` field that replaces `extra`.
- db_obj[meta] and db_obj[extra] are the database representations of those node fields.

Getting objects from the database (API/conductor < DB)

Both ironic-api and ironic-conductor services read values from the database. These values are converted to IronicObjects via the existing method `IronicObject._from_db_object(context, obj, db_object)`. This method will be changed so that the IronicObject will be in the latest version, even if it was in an older version in the database. This is done regardless of the service being pinned or not.

Note that if an object is converted to a later version, that IronicObject will retain any changes resulting from that conversion (in case the object later gets saved in the latest version).

For example, if the node in the database is in version 1.14 and has db_obj[extra] set:

- a `FromVer` service will get a Node with node.extra = db_obj[extra] (and no knowledge of node.meta since it doesnt exist).
- a `ToVer` service (pinned or unpinned), will get a Node with:
  - node.meta = db_obj[extra]
  - node.extra = None
  - node._changed_fields = [meta, extra]

Saving objects to the database (API/conductor > DB)

The version used for saving IronicObjects to the database is determined as follows:

- for an unpinned service, the object will be saved in its latest version. Since objects are always in their latest version, no conversions are needed.
- for a pinned service, the object will be saved in its pinned version. Since objects are always in their latest version, the object will need to be converted to the pinned version before being saved.

The new method `IronicObject.do_version_changes_for_db()` will handle this logic, returning a dictionary of changed fields and their new values (similar to the existing `oslo.versionedobjects.VersionedObjectobj.obj_get_changes()`). Since we do not keep track internally, of the database version of an object, the objects `version` field will always be part of these changes.

The *Rolling upgrade process* (at step 6.1) ensures that by the time an object can be saved in its latest version, all services are running the newer release (although some may still be pinned) and can handle the latest object versions.

An interesting situation can occur when the services are as described in step 6.1. It is possible for an IronicObject to be saved in a newer version and subsequently get saved in an older version. For example, a `ToVer` unpinned conductor might save a node in version 1.5. A subsequent request may cause a `ToVer` pinned conductor to replace and save the same node in version 1.4!

Sending objects via RPC (API/conductor -> RPC)

When a service makes an RPC request, any IronicObjects that are sent as part of that request are serialized into entities or primitives (via `oslo.versionedobjects.VersionedObjectSerializer.serialize_entity()`). The version used for objects being serialized is as follows:

- for an unpinned service, the object will be serialized in its latest version. Since objects are always in their latest version, no conversions are needed.
- for a pinned service, the object will be serialized in its pinned version. Since objects are always in their latest version, the object will need to be converted to the pinned version before being serialized. The converted object will include changes that resulted from the conversion; this is needed so that the service at the other end of the RPC request has the necessary information if that object will be saved to the database.

The `IronicObjectSerializer.serialize_entity()` method will be modified to do any IronicObject conversions.

Receiving objects via RPC (API/conductor <- RPC)

When a service receives an RPC request, any entities that are part of the request need to be deserialized (via `oslo.versionedobjects.VersionedObjectSerializer.deserialize_entity()`). For entities that represent IronicObjects, we want the deserialization process to result in IronicObjects that are in their latest version, regardless of the version they were sent in and regardless of whether the receiving service is pinned or not. Again, any objects that are converted will retain the changes that resulted from the conversion, useful if that object is later saved to the database.

The deserialization method invokes `VersionedObjectSerializer._process_object()` to deserialize and get the IronicObject. We will add `IronicObjectSerializer._process_object()` to convert the IronicObject to its latest version.

For example, a `FromVer` ironic-api could issue an update_node() RPC request with a node in version 1.4, where node.extra was changed (so node._changed_fields = [extra]). This node will be serialized in version 1.4. The receiving `ToVer` pinned ironic-conductor deserializes it and converts it to version 1.5. The resulting node will have node.meta set (to the changed value from node.extra in v1.4), node.extra = None, and node._changed_fields = [meta, extra].

Alternatives

A cold upgrade can be done, but it means the ironic services will not be available during the upgrade, which may be time consuming.

Instead of having the services always treat objects in their latest versions, a different design could be used, for example, where pinned services treat their objects in their pinned versions. However, after some experimentation, this proved to have more (corner) cases to consider and was more difficult to understand. This approach would make it harder to maintain and trouble-shoot in the future, assuming reviewers would be able to agree that it worked in the first place!

What if we changed the ironic-api service, so that it had read-only access to the DB and all writes would go via the ironic-conductor service. Would that simplify the changes needed to support rolling upgrades?

Perhaps; perhaps not. (Although this author thinks it would be better, regardless, to have all writes being done by the conductor.) With or without this change, we need to ensure that objects are not saved in a newer version (i.e., that is newer than the version in the older release) until all services are running with the new release step 5.2 of the *Rolling upgrade process*. The solution described in this document has objects being saved in their newest versions starting in step 6.1, because it seemed conceptually easy to understand if we save objects in their latest versions only when a service is unpinned. Wed need a similar mechanism regardless.

Of course, there are probably other ways to handle this, like having all services register what versions they are running in the database and leveraging that data somehow. Dmitry Tantsur mused about whether some remote synchronization stuff (e.g. etcd) could be used for services to be aware of the upgrade process.

Ideally, ironic would use some OpenStack-preferred way to implement rolling upgrades but that doesnt seem to exist, so this tries to leverage the work that nova did.

Data model impact

A DB migration policy is adopted and introduced above in *New DB model change policy*.

A new `version` column will be added to all the database tables of the IronicObject objects. Its value will be the version of the object that is saved in the database.

State Machine Impact

None

REST API impact

There is no change to the REST API itself.

During the rolling upgrade process, the API services may run in different versions at the same time. Both API service versions should be compatible with the python-ironicclient library from the older release (we already use microversions to guarantee this). New API functionality is available everywhere only after completing the upgrade process, which includes unpinning of the internal RPC communication versions on all ironic services. Therefore, until the upgrade is completed, API requests should not be issued for new functionality (i.e., with new API micro versions) since there is no guarantee that they will work properly.

As a future enhancement (outside the scope of this specification), we could disallow requests for new functionality while the API service is pinned.

Client (CLI) impact

None

ironic CLI

None

openstack baremetal CLI

None

RPC API impact

There is no change to the RPC API itself, although changes are needed for supporting different RPC API versions. `version_cap` will be set to the pinned version (the previous release) to make compatible RPC calls of the previous release.

Developers should keep this in mind when changing an existing RPC API call.

Driver API impact

None

Nova driver impact

Since there is no change to the REST API, there is no need to change the nova driver. Ironic should be upgraded before nova, and nova calls ironic using a specific micro version that will still be supported in the upgraded ironic. Thus everything will work fine without any changes to the nova driver.

Ramdisk impact

None

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

Operations which involve migration of data may take longer during the upgrade. Each such change should be mentioned in the release notes provided with the impacted release.

Other deployer impact

During the rolling upgrade, the deployer will use the new configuration option `[DEFAULT]/pin_release_version` to pin and unpin the RPC and IronicObject versions used by the services, as described above in *RPC and object version pinning*.

This specification doesnt address trying to stop/roll back to a previous release.

Developer impact

The code contribution guide[11] will be updated to describe what a developer needs to know and follow. It will mention:

[11] http://docs.openstack.org/developer/ironic/dev/code-contribution-guide.html#live-upgrade-related-concerns

- before a release is cut, manually add the mapping of release (named or sem-versioned) with the associated RPC and Object versions
- the new DB model change policy
- the contribution guide will point to design documentation, as it relates to how new features should be implemented in accordance with rolling upgrades

Implementation

Assignee(s)

Primary assignee:

- xek
- rloo

Other contributors:

- mario-villaplana-j (documentation)

Work Items

1. Add new configuration option `[DEFAULT]/pin_release_version` and RPC/Object version mappings to releases.
2. Make Objects compatible with a previous version and handle the interaction with DB and services.
3. Make IronicObjectSerializer downgrade objects when passing via RPC.
4. Add tests.
5. Add documentation and pointers for RPC and oslo objects versioning.
6. Add documentation for the new DB model change policy.
7. Add admin documentation for operators, describing how to do a rolling upgrade, including the order in which all related services should be upgraded.

Dependencies

- Needs the new command **ironic-dbsync online_data_migration**[Page 439, 12].
- Needs multi-node grenade CI working.

Testing

- unit tests
- multi-node grenade CI. This tests that the rolling upgrade process continues to work from `fromVer` to `toVer`.
  - grenade does a full upgrade, without pinning. This tests old API/conductor and new API/conductor unpinned. We wont run online data migrations, so the new services will be reading the old format of the data.
  - grenade multi-node will be running old API/conductor on the subnode, which wont be upgraded. The primary will have conductor only, which does get upgraded, but is pinned. This

tests old API + old data, with 1. old conductor, and 2. new conductor with a pin. Tests are run pre/post-upgrade.

- We could also move the API to the primary node, upgrade it, pinned, to test new API code with the pin. Since all the object translation happens at the objects layer, this may not be needed since it may not test much that hasnt already been exercised in the conductor.
- The above two tests could be merged, if grenade can be set-up to stop the old API service on the subnode and start it on the upgraded primary after doing the first test.

Tests should cover the use cases described in *Rolling upgrades between releases* as much as possible.

Upgrades and Backwards Compatibility

None; there is no change to the REST API or Driver API.

Documentation Impact

Documentation will be added for:

- deployers. This will describe the rolling upgrade process and the steps they will need to take. This should be documented or linked from the upgrade guide[Page 437, 13].
- developers. This will describe what a developer needs to know so that the rolling upgrade process continues to work. This includes documentation on RPC and oslo objects versioning, as well as DB model change policy.

References

5.9.2 9.0

Boot from Volume - Reference Drivers

https://bugs.launchpad.net/ironic/+bug/1559691

In order to support booting ironic nodes from a storage device that is hosted and/or controlled remotely, such as via a SAN or cinder iSCSI target, changes are required to ironic in order to help facilitate and enable drivers to leverage the remote storage as a boot volume.

Problem description

- Presently ironic has no support in the reference drivers to boot a node from a remote volume.
- Due to the lack of this support, users capabilities and usability are limited. Providing this functionality allows deployments to leverage greater capabilities.

Boot from Volume scenarios

This specification is broken into multiple portions in order to facilitate a phased implementation of support. While it would appear that these changes should be limited to the boot and deployment interfaces, they ultimately require substantial substrate changes to provide necessary functionality.

As there is a large variety of potential connection methods and configurations, the scenarios we have identified appear suitable for implementation in a reference driver. We are primarily focused on the use of cinder, but intend to make the storage substrate pluggable and generic.

Not covered as part of this specification is handling of support for MultiPath IO, sometimes referred to as MPIO. We feel this is largely outside of Ironics scope at this point in time.

Additionally not covered is the requirement to use UEFI HTTP boot support, which was set forth during the ironic mitaka midcycle. Using UEFI HTTP boot would simply load the iPXE UEFI boot loader, and would thus be a redundant capability compared with existing capabilities.

Below are various boot from volume scenarios. These scenarios are intended to provide a clear definition of conditions and requirements.

Scenario 1 - iPXE boot from iSCSI volume

Conditions:

- Node boot defaults to network booting via iPXE controlled by the ironic conductor. This is similar to the PXE driver netboot option.
- Metadata service use is expected as configuration drive support is not feasible at this time.
- **Configuration drive is not supported.**
  - For context: This is a limitation as a result of the minimum new volume and new volume extension sizes in cinder. This may be a capability at a later point in time, but is out of scope for an initial implementation.
- iPXE version sufficient to support boot from iSCSI target.
- Operating system is already deployed to the intended iSCSI boot volume via external means.
- **Tenant network isolation is unsupported due to the need for iPXE boot.**
  - Potentially in the future, a framework or proxy could be created to enable tenant network isolation, however that is considered out of scope at this time.
- Node is configured by the operator with the `node.properties['capabilities']` setting of `iscsi_boot` set to a value of `true` for node validation to succeed.
- Node has a defined volume target with a `boot_index` value of `0` in the volume_targets table.

Requirements:

- **Storage driver interface**
  - A storage driver/provider interface is needed in order to support recovery of systems in the event of a failure or change in hardware configuration.
- iPXE template and generation logic changes.
- Implementation of substrate logic in the deploy and PXE driver modules to appropriately handle booting a node in a boot from volume scenario.

Scenario 2 - iPXE boot from Fibre Channel over Ethernet (FCoE)

Conditions:

- Node boot defaults to network booting via iPXE controlled by the ironic conductor. Similar to the PXE driver netboot option.
- Metadata service use is expected.
- Configuration drive is not supported.
- Operating system is already deployed to the intended boot volume via external means.
- Tenant network isolation is unsupported due to iPXE requirements.
- Node is configured with the `node.properties['capabilities']` setting of `fiberchannel_boot` set to a value of `true` for node validation to succeed.
- Node has a defined volume target with a `boot_index` value of `0` in the volume_targets table.

Requirements:

- *Scenario 1 - iPXE boot from iSCSI volume* implemented
- Additional logic for iPXE template generation in the FCoE use case

Scenario 3 - Boot via hardware Host Bus Adapter (HBA) to volume

Scenario involves boot to a local volume, however, the volume is expected to be pre-populated with a bootable operating system. No volume deployment operation is expected.

This scenario is based upon the expectation that the environment and the nodes firmware settings are such that it is capable of booting directly to a presented volume via the HBA once powered on.

Conditions:

- Metadata service use is expected.
- Configuration drive support is unavailable.
- Node boot defaults to block storage device.
- Tenant network isolation is supported as this scenario does not require iPXE booting for normal operation of the host.
- Operating system is already deployed to the intended boot volume via external means.
- Infrastructure and HBA BIOS configuration is such that the node will boot to what is expected to be the Logical Unit Number (LUN) offered as the boot volume.
- Node is configured with the `node.properties['capabilities']` setting of `hba_boot` set to a value of `true` for node validation to succeed.
- Node has a defined volume target with a `boot_index` value of `0`.

Requirements:

- Creation of driver deploy method functionality that under normal circumstances performs a no-op, and defaults the next boot state to local disk when the user requests a node to be deployed.

Scenario 4 - Boot via Host Bus Adapter (HBA) to volume with image deployment

Scenario involves boot to a local volume where the volume needs to be written out by Ironics mechanisms.

Conditions:

- Metadata service is not required.
- Configuration drive is supportable.
- Node boots to block storage device.
- Tenant network isolation is supported as this scenario does not require iPXE booting for normal operation of the host.
- Infrastructure and HBA BIOS configuration is such that the node will boot to what is expected to be the LUN offered as the boot volume.
- Node is configured with the `node.properties['capabilities']` setting of `hba_boot` set to a value of `true`.
- Node is configured with the `node.instance_info['force_deployment']` parameter set to ``true`.

Requirements:

- This method is expected to be essentially identical to the scenario defined in *Scenario 3 - Boot via hardware Host Bus Adapter (HBA) to volume*, however via the inclusion of default logic that only invokes deploy phase when explicitly requested for boot from a volume.

Scenario 5 - iPXE boot to NFS root volume

Scenario involves use of a kernel and ramdisk image hosted by the conductor host in order to enable the node to boot via iPXE with sufficient command line parameters to enable the root volume to attach during the boot sequence.

This is a logical progression given that users have indicated that they have enabled similar boot functionality downstream.

Conditions:

- Metadata service use expected.
- Configuration drive support is unavailable.
- Node boot defaults to iPXE.
- Node boot utilizes kernel and ramdisk hosted by the conductor.
- Operating System is already deployed to the intended boot volume via external means.
- Tenant network isolation is unsupported due to iPXE need.

- Node is configured with `node.properties['capabilities']` setting of `nfs_boot` set to a value of true coupled with an `instance_info` setting of `nfs_root` which provides nfs root connection information.
- Kernel and ramdisk utilized support the nfsroot kernel command line option.

Potential future capabilities

These are some additional items that it might make sense to develop later on after reference driver implementation has been completed, however these are out of scope of the existing specification.

- Boot the agent from a remote volume to facilitate a deployment.
- Creation of a deployment framework to allow IPA to potentially apply HBA settings.
- Multipath IO configuration handling and potentially passing.
- Configuration drives support.
- Support for tenant network isolation boot scenarios.

Proposed change

In order to support an initial feature set, i.e. Scenarios 1-4, for boot from volume, we propose the following:

- Implementation of a basic capability concept via a helper method that will allow a driver/provider capability to be checked by another portion of the code base, returning false if the capabilities definition is missing. Example:

```
utils.check_capability(
    task.driver.boot.capabilities, 'iscsi_volume_boot')
utils.check_compatibility(
    task.driver.deploy.capabilities, 'iscsi_volume_deploy')
```

  This would be implemented via a list of capability tags to each main level interface class, as required in order to guard against invalid configurations.

- Implementation of logic in the existing reference driver deploy validate methods to fail the validation of any node that has volume storage configured when the driver that a node is configured with lacks any such support as identified by the previously noted capability interface. This is to help ensure that such nodes are not accidentally provisioned with erroneous user expectations.
- Updating of the `agent.AgentDeploy` and `iscsi_deploy.ISCSIDeploy` driver logic to support skipping deployment of a node if a storage interface provider is defined, volume information exists for the volume, and the volume has an index of `0` indicating it is the boot volume. In essence, this means that if the node is defined to boot from a remote volume, that the driver.deploy.deploy method should immediately return DEPLOYDONE as any network booting configuration, if applicable, would have to be written out via driver.deploy.prepare method. Example:

```
if (task.node.storage_interface is not None and
        not task.driver.storage.should_write_image(task)):
```

  Additionally, validation logic will need to be updated in the deploy drivers to pass specific checks of instance_info fields that do not apply with the case of booting from a volume.

- **Creation of a storage provider interface:**
  - Similar in composition to the network provider interface, where a default will result in a provider interface that performs `no-op` behavior, while exposing an empty set of storage capabilities.
  - A node level `storage_interface` setting with default value, similar to the Add network interface to base driver class change, to define if a node is to utilize the storage interface along with the provider that is to be utilized. This is intended to align with the Driver composition reform specification.
  - Initial and reference storage driver to be based upon cinder, leveraging `python-cinderclient` for the REST API interface, to support the following fundamental operations detailed below.
    - `detach_volumes` - Initially implemented to enumerate through known attached volumes and remove the storage attachments.
    - `attach_volumes` - Initially implemented to enumerate through known configured volumes and issue storage attachment requests. In the case of the cinder driver, we will reserve the volumes in our configuration, initialize connections to the volumes meanwhile supplying current initiator configuration, then trigger the cinder volume attach call to update the database configuration. Additionally, we will update the volume metadata to allow for easy user identification of volumes that are used for ironic nodes by recording information to allow for reconciliation of nodes that are powered off with detached volume.
    - `validate` - Initially implemented to validate that sufficient configuration is present in the configuration to allow for the functionality to operate. This validation will be skipped if no volumes are defined.
    - `should_write_image` - Provides an interface to allow the other components to ask the storage driver if the image should be written to disk. This allows all of the logic to be housed with the storage interface.
- Updating of the `pxe.PXEBoot` driver logic to support the creation of the appropriate iPXE configurations for booting from the various boot scenarios if the `volume_target` information is defined, iPXE is enabled, and a storage provider is available.
- Updating of the `pxe.PXEBoot` validate interface to leverage a helper method when a storage provider and boot volume is defined in the node configuration, to validate that the capabilities, initiator configuration, and volume configuration are in alignment for the specified volume/path type. Note that the bulk of this logic should reside in a `deploy_utils` method that can be re-used by other drivers moving forward.
- Updating of the conductor `utils.node_power_action` logic to enable the storage provider (defined by the node.storage_interface setting) to be called to permit volume attachment and detachment for the node and thus update any storage configuration if necessary.
- Addition of a helper method that sets and returns, if not already set, the node.instance_infos boot_option parameter based upon the hardware configuration, and supplied volume configuration information, enabling matching and identification of what the next appropriate step is for each scenario.
- Updating of the conductor `_do_node_tear_down` method to call the storage provider detach_volumes method and purge volume information if not already implemented.

  - At the beginning and completion of the storage detachment interaction, a notification shall be generated to allow visibility into if the process is successfully completed.

- Updating the iPXE template logic to support the creation of the various file formats required for Scenarios 1, 2, 5. See: IPXE sanhook command, IPXE san connection URLs and nfsroot kernel command line option.

As previously noted, each scenario will be submitted separately as incremental additions to ironic.

In order to support scenario 4, the deploy driver will need to understand how to deploy to a system with such configuration.

- Updating of the deploy driver to enable the logic added for scenarios 1-3 to be bypassed using a `force_deployment` parameter which should be removed from the nodes instance_info prior to the node reaching the active state. This, in effect, would cause ironic to support a deployment operation when the supplied volume information is normally expected to have a valid Operating System and boot loader already in place.
- Agent will need to be informed of the desired volume for the boot volume, and, if supplied to the target, connection information. The appropriate information should be passed in using Root device hints, specifically setting a WWN or volume serial number if available.

In order to support scenario 5:

- Scenario 3 must be implemented. This is anticipated to largely be an alternative path in the iPXE template where the previously defined settings cause the on-disk PXE booting template to boot the node from the NFS root.

Later potential improvements above and beyond this initial specification:

- Creation of logic to allow Ironic users to leverage the storage provider to request a volume for their node. Such functionality would require ironic to deploy the OS image, and should be covered by a separate specification.

Alternatives

An alternative could be to simply not develop this functionality and to encourage downstream consumers to independently develop similar tooling to meet their specific environments needs. That being said, both options are unappealing from the standpoint of wishing to grow and enhance ironic.

Data model impact

A *storage_interface* field will be added to the node object which will be mapped to the appropriate storage driver.

State Machine Impact

None

REST API impact

As the node storage driver will be selectable by the operator, it will need to be concealed from older API clients, which will necessitate a microversion update once the functionality is present in the API.

Client (CLI) impact

ironic CLI

None. All expected CLI updates are expected to be part of the specification covering information storage, Add volume connection information for Ironic nodes.

openstack baremetal CLI

None

RPC API impact

None

Driver API impact

The first change is the introduction of a list of supported advanced driver features defined by the deploy and boot driver classes, known as capabilities, that allow for other driver components to become aware of what functionality is present in a neighboring driver interface/composition.

The second change is the introduction of a new *storage_interface* that will be mapped to a selectable storage driver of the available storage drivers.

Within this storage interface, new methods will be defined to support expected storage operations. Below are the methods that are anticipated to be added and publicly available from drivers:

```
def attach_volumes(self, task):
"""Informs the storage subsystem to detach all volumes for the node."""

def detach_volumes(self, task):
"""Informs the storage subsystem to detach all volumes for the node."""

def validate(self, task):
"""Validate that the storage driver has appropriate configuration."""

def should_write_image(self, task):
"""Determines if deploy should perform the image write-out."""
```

Nova driver impact

Impact to the nova driver is covered in a separate specification Nova Specs - Ironic - Boot from Cinder volume. As the driver will be available as an opt-in change, we expect no negative impact on behavior.

Ramdisk impact

While we should be able to derive the intended root volume and pass an appropriate root hint if necessary in order to facilitate a deployment as part of *Scenario 4 - Boot via Host Bus Adapter (HBA) to volume with image deployment* , the IPA ramdisk should likely have functionality added in the form of a HardwareManager in order to support MutliPath IO. That being said MPIO is out of scope for this specification.

Security impact

Overall, the storage driver, in this case, cinder, will need to utilize credentials that are already populated in the configuration for keystone to connect to cinder to obtain and update mapping information for volumes.

Scenarios 1-2 and 5 are designed such that the tenant machine is able to reach the various services being offered up by whatever the volume driver is set to leverage over the nodes default network configuration. As a result of the need to network boot, the flat network topology is required along with access controls such that the nodes are able to reach the services storage volumes.

The more secure alternative are the drivers representing scenarios 3 and 4 as this configuration ultimately requires a separate storage infrastructure. This case will allow for tenant network isolation of deployed nodes.

Other end user impact

This functionality may require additional knowledge to be conveyed to the ironic-webclient and ironic-ui sub-projects, however that will need to be assessed at the time of implementation as they are under active development.

Scalability impact

This is a feature that would be opted into use by an operator. In the case where it is active, additional calls to the backend storage interface may have a performance impact depending upon architecture of the deployment.

Performance Impact

For each node under ironics care that we believe has volumes, we need to query storage manager, presumably cinder based on this implementation, and attach/detach volumes during intentional user drive power operations. This may extend the call to power-on a node after deployment, or potentially prevent power-up if the attachment cannot be made.

Other deployer impact

Deployers wishing to use these drivers will naturally need to add the `cinder` storage interface to the `enable_storage_interfaces` list.

A default storage configuration driver will be set to `noop` which will prevent any storage related code from being invoked until the operator explicitly chooses to enable this support.

Based on the proposed driver configuration, we can expect two additional sections in the conductor configuration file:

```
[DEFAULT]
enabled_storage_interfaces = <None> # Defaults to none and is a list of the
available drivers.

[cinder]
url = http://api.address:port
url_timeout = 30
retries = 3
auth_strategy = <noauth|keystone>
toggle_attachments_with_power = true
```

Developer impact

This will increase the overall complexity and set of capabilities that ironic offers. Driver developers wishing to implement driver specific functionality should expect certain substrate operations to occur, and attempt to leverage the substrate set forth in this specification and the Add volume connection information for Ironic nodes specification.

Implementation

Assignee(s)

Primary assignee:
juliaashleykreger

Other contributors:
blakec shiina-hironori

Work Items

The breakdown of the proposed changes, when combined with the underlying scenarios helps convey the varying work items. That being said, this functionality will take some time to land.

Dependencies

Logic will need to be implemented in IPA to handle the scenario when no disks are detected. `cleaning` and `inspection` operations should be able to be executed upon hardware nodes that have no local disk storage.

Additionally in IPA, as a soft dependency, logic MAY be required to better handle directly attached volume selection when multipathing is present. This will require its own specification or well-defined and validated plan as IPA cannot expect OS multipathing support to handle MPIO scenarios in all cases, or to even be present.

Implementation of Add volume connection information for Ironic nodes. This specification should not be entirely dependent upon the implementation of the Nova Specs - Ironic - Boot from Cinder volume specification.

A soft dependency exists for the Driver composition reform specification in that these two efforts are anticipated to be worked in parallel, and this implementation effort should appropriately incorporate functionality as the functionality for the Driver composition reform specification begins to become available.

Testing

The level of full stack functional and integration tests is a topic that requires further discussion in the ironic community. An initial case for a gate test could be where an ironic deployment boots from a Cinder volume, which a tempest test could orchestrate.

Scenarios 3 and 4 are the most difficult to test as they have detached infrastructure expectations outside of our direct control. However, we may find that the base overlay is sufficient to test with unit tests due to what will ultimately be significant underlying common paths.

Upgrades and Backwards Compatibility

As this feature set is being created as a new set of capabilities within the reference drivers and their capability, no compatibility issues are expected as the API field additions related to this specification will be hidden from an API client that does not request the appropriate API version.

A database migration step will be added to create the `storage_interface` node database field. The initial value for this field will be None, and there will be no implied default set as an operator must choose to enable a storage interface in their environment.

Documentation Impact

Documentation will need to be updated detailing the new driver and the related use scenarios so an operator can comprehend what options the driver(s) provide and how they can fit into their use cases. Additional caveats regarding long term network booting of hosts should be explicitly stated as part of this work.

It is expected that this specification will be further refined during development of this functionality in order to raise and document any new findings at a technical level.

References

Relevant specifications:

- Add volume connection information for Ironic nodes
- Nova Specs - Ironic - Boot from Cinder volume

Mitaka midcycle etherpad:

- ironic mitaka midcycle

Dynamic port groups

https://bugs.launchpad.net/ironic/+bug/1652630

In the Ocata release, Ironic added support for grouping ports into port groups. The administrator also has the ability to specify `mode` and `properties` of a port group by using appropriate port group fields. However the administrator is still required to pre-create a port group on the ToR (Top of Rack) switch manually. Or find some other way to sync portgroup settings between Ironic and the ML2 driver.

Problem description

While Ironic provides the ability to create port groups with different configurations (mode and properties), port groups still have to be pre-created on ToR manually as we do not pass port group details to ML2 drivers. To make port groups creation `dynamic` Ironic should pass port group settings to Neutron ML2 drivers. The ML2 driver is responsible for port group configuration (creation/deletion) on ToR switch during deployment.

This spec does not cover end-user interface for specifying port group configuration when doing `nova boot` at this moment. It can be extended when community has some agreement on it.

Proposed change

Start passing port group information to Neutron ML2 drivers via the Neutron port `binding:profile` field. Appropriate Neutron ML2 drivers will use that information to create port group dynamically on the ToR switch.

> **Note**
>
> We cannot use existing `binding:profile` `local_link_information` key as it is a list with port details, where `switch_id` and `port_id` are mandatory keys. For portgroup object those keys are not required as portgroup is virtual interface and might be spread across different switches. For example MLAG configuration.

Binding profile data structure

- Introduce new `local_group_information` dictionary that stores portgroups information.
- Reuse existing `local_link_information` for port objects only

A JSON example of `binding:profile` with `local_link_information` reuse:

```
"binding:profile": {
    'local_link_information': [
        {
            'switch_info': 'tor-switch0',
            'port_id': 'Gig0/1'
            'switch_id': 'aa:bb:cc:dd:ee:ff'
        },
        {
            'switch_info': 'tor-switch0',
            'port_id': 'Gig0/2',
            'switch_id': 'aa:bb:cc:dd:ee:ff'
        }
    ],
    'local_group_information': {
            'id': '51a9642b-1414-4bd6-9a92-1320ddc55a63',
            'name': 'PortGroup0',
            'bond_mode': 'active-backup',
            'bond_properties': {
                'bond_xmit_hash_policy': 'layer3+4',
                'bond_miimon': 100,
            }
        },
}
```

The data types:

Field Name	Description
id	The UUID of Ironic portgroup object
name	The name of the ironic port group
bond_mode	Ironic portgroup mode
bond_properties	Ironic portgroup properties
switch_info	The hostname of the switch
port_id	The identifier of the port on the switch
switch_id	The identifier of the switch, ie mac address

Note

It is recommended to pick `bond_mode` and keys/values for `bond_properties` from the[1] as they will be used by user OS.

Alternatives

- Use port groups in the static fashion when administrator pre-creates port group on ToR switch.
- If ML2 driver supports port group creation, make sure that port group properties in Ironic and ML2 are the same.

Data model impact

None.

State Machine Impact

None.

REST API impact

None.

Client (CLI) impact

None.

RPC API impact

None.

Driver API impact

None.

[1] *Linux kernel bond*: https://www.kernel.org/doc/Documentation/networking/bonding.txt

Nova driver impact

None.

Ramdisk impact

None.

Security impact

None.

Other end user impact

None.

Scalability impact

None.

Performance Impact

None.

Other deployer impact

No need to pre-create port group at the ToR switch. Only need to specify port group configuration at the Ironic portgroup object.

Developer impact

Out of tree network interfaces should be updated to pass `portgroup.mode` and `portgroup.properties` with `links` array in Neutron port `binding:profile` field. Vendors are responsible to deal with `links` to support dynamic port groups.

Implementation

Assignee(s)

Primary assignee:
vsaienko <vsaienko@mirantis.com>

Work Items

- Update `neutron` network interface to pass data structure described in *Binding profile data structure* to Neutron.
- Add dynamic port group support to networking-generic-switch
- Update tempest with appropriate tests.

Dependencies

Dynamic portgroup support is dependent on Neutron ML2 driver functionality being developed to deal with `links` array in `binding:profile` field.

Testing

- Add dynamic port group support to networking-generic-switch
- Update tempest with appropriate tests.

Upgrades and Backwards Compatibility

Backward compatibility is retained as Ironic will still pass `local_link_information` in Neutron port `binding:profile` field.

Documentation Impact

This feature will be fully documented.

References

Physical Network Awareness

https://bugs.launchpad.net/ironic/+bug/1666009

Ironic ports and portgroups correspond to the physical NICs that are available on baremetal nodes. Neutron ports are logical ports attached to tenant and provider networks. When booting baremetal nodes in a system with multiple physical networks, the mapping between logical and physical ports must account for the physical network connectivity of the system.

This feature aims to make ironic aware of the `physical_network` attribute of neutron networks and network segments.

Problem description

Physical Networks

A neutron port on a provider network or network segment is associated with a physical network. The physical network concept is used to specify the physical connectivity of networks and network segments in the system.

There are many reasons why an operator might use multiple physical networks, including:

- routed provider networks[1]
- security, through physical segregation of traffic
- redundancy, through independence of multiple networks
- traffic isolation (e.g. storage vs. application)
- different characteristics (bandwidth, latency, reliability)
- different technologies (Ethernet, Infiniband, Omnipath, etc.)

[1] Neutron routed networks

It is possible that some or all nodes may be connected to more than one physical network. OpenStack adoption is growing rapidly in the scientific computing community, an area in which the use of nodes connected to multiple physical networks is prevalent.

For context, we provide some examples of how the physical network attribute is used in other OpenStack components. The neutron ML2 Open vSwitch agent is made physical network-aware via the `[OVS] bridge_mappings` configuration option. This option maps physical network names to Open vSwitch bridges that have a physical network interface as a port. This option is used for flat and VLAN network types, and is taken into consideration when binding ports to network segments in the neutron server ML2 driver. This acts both to ensure that physical connectivity allows for the requested logical topology, and also supports hosts being connected to multiple physical networks.

A similar mapping exists in nova for pass-through of PCI physical devices or SR-IOV virtual functions via the `[DEFAULT] pci_passthrough_whitelist` configuration option.

Physical Networks in Ironic

An ironic port is connected to a physical network. As portgroups are a layer-2 construct, all ports in a portgroup should be in the same physical network. If a neutron port is mapped to a port or portgroup and is attached to a neutron network or network segment on a different physical network, there will be no connectivity between the bare metal nodes NIC and other neutron ports on the network. This is perhaps most obvious when it results in a failure to acquire a DHCP lease for the interface.

The mapping between logical ports in neutron and physical ports and portgroups in ironic has always been somewhat unpredictable[2]. The ironic-neutron integration work added support for local link information for ports[3]. In the interface attach/detach API work[4] ironic moved the responsibility for attaching virtual interfaces from nova to ironic. In both of these features physical network-awareness was seen as out of scope.

Currently when a virtual interface is attached to a node, the procedure used to map it to an ironic port or portgroup is roughly as follows:

- if there is a free portgroup, select one
- else if there is a free port with PXE enabled, select one
- else if there is a free port with PXE disabled, select one
- else fail

This algorithm takes no account of the physical network to which the ports and portgroups are connected, and consequently can result in invalid mappings. Further, there is currently no standard way in ironic to specify the physical network to which a port or portgroup is connected.

Provisioning and Cleaning Networks

The ironic pluggable network provider[5] work added support for attaching nodes to dedicated provisioning and cleaning networks during provisioning and cleaning operations respectively. Currently, all ironic ports and portgroups with PXE enabled are attached to the provisioning or cleaning network. While this ensures that the node has a port on the provisioning or cleaning physical network, it may result in unnecessary neutron ports being created if some of the ironic ports or portgroups are connected to a

[2] ports cannot be mapped to networks
[3] Ironic neutron integration
[4] interface attach/detach API
[5] pluggable network providers

different physical network. In practice this can be avoided by disabling PXE on ports where it is not required.

Scheduling

In a system with multiple physical networks where not all nodes are connected to every physical network, it becomes possible for a user to request a logical network topology that cannot be realised. Without awareness of the physical networks that each ironic node is connected to, nova cannot reliably schedule instances to ironic nodes. This problem is considered beyond the scope of this spec.

Proposed change

There are four parts to the proposed solution to this problem.

1. It must be possible to specify the physical network to which an ironic port is connected.
2. The ironic network interfaces must account for the physical network of an ironic nodes ports and portgroups when attaching tenant virtual interfaces.
3. When attaching a node to a provisioning or cleaning network, neutron ports should be created only for ironic ports and portgroups on the same physical network as the provisioning or cleaning network.
4. The binding profiles of attached neutron ports should be updated with the physical network of the ironic port or portgroup to which the port was mapped.

Tagging Ports

There are a few options for how ports might be tagged with a physical network:

1. a new attribute of the `local_link_connection` field
2. a new attribute of the `extra` field
3. a new first class field

Reusing an existing field would certainly be easier to implement than adding a new one, but OpenStack history has frequently shown that buried fields often end up needing to become first class citizens. A relevant example here is the `provider:physical_network` extension field on neutron networks, which later became a first class field on network segments[Page 461, 1]. Further, if ironic intends to support physical network-aware scheduling in future, the ability to efficiently filter ports by their physical network may be advantageous. This spec therefore proposes to add a new first class `physical_network` field to ironics `Port` object. For backwards compatibility, this field will be optional.

The process of mapping physical networks to ironic ports is out of scope for ironic. This could be done either through a manual procedure or through an automated process using information gathered during a hardware introspection process. For example, if using ironic inspector to perform introspection it would be possible to create an introspection plugin[6] that maps switch IDs discovered via LLDP to physical networks.

[6] introspection plugins

Portgroups

The physical network of a portgroup will be determined through the physical network of its constituent ports. All ports in the portgroup must have the same physical network, and this will be enforced in the ironic API when creating and updating ports.

This has the unfortunate consequence of making it rather unwieldy to update the physical network of the ports in a portgroup, since the ports must be removed from the portgroup while their physical network is updated. This may be improved upon in future through the use of a virtual physical network field in the portgroups API that allows simultaneous update of the physical network field of all the ports in the group.

Mapping Logical Ports to Physical Ports

In order to account for physical network connectivity, the virtual interface attachment algorithm must determine the physical networks that the neutron port being attached can be bound to. This information is available via the neutron API as the `physical_network` field on network segments in the ports network or as `provider:physical_network` on the ports network.

The virtual interface attachment mapping algorithm will be modified to the use the following set of criteria listed in order of descending priority:

1. reject ports and portgroups with a non-null physical network that is different than all of the networks physical networks
2. prefer ports and portgroups with a non-null physical network to ports with a null physical network
3. prefer portgroups to ports
4. prefer ports with PXE enabled to ports with PXE disabled

This algorithm provides backwards compatibility for environments in which the port(s) and/or portgroup(s) associated with the ironic node do not have a `physical_network` property configured.

Provisioning and Cleaning Networks

In ironic network drivers that support network flipping for provisioning and cleaning operations, we will create neutron ports only for those ironic ports and portgroups that have PXE enabled and are on the same physical network as the provisioning or cleaning network in question, or do not have a physical network specified.

Neutron Port Binding Profiles

When attaching virtual interfaces to physical or virtual functions of PCI network devices, nova sets a `physical_network` attribute in the `binding:profile` field of the neutron port. Further research is required to determine what effect it would have if ironic were to do the same.

Alternatives

We could continue to use an unpredictable mapping between logical ports and physical ports. This limits the use of ironic to environments in which there is only one physical network.

We could continue with the existing mapping algorithm in ironic but provide neutron with the information required to determine whether a mapping is valid from the `local_link_connection` binding information. Ironic would then be modified to retry interface attachment with a different neutron port if neutron determined the mapping to be invalid. This method would be inefficient due to the retries necessary.

We could avoid the need to tag ironic ports with a physical network by providing a mechanism to map from the information in their `local_link_connection` fields to a physical network. This would require either an addition to ironics data model to support Switch objects or a new neutron API providing a lookup from switch ID to physical network.

Data model impact

A new `physical_network` field will be added to Port object. In neutron the `Segment` objects `physical_network` field is defined as `sqlalchemy.String(64)`, so the same will be used in ironic.

State Machine Impact

None

REST API impact

The port REST API will be modified to support the new `physical_network` field. The field will be readable by users with the baremetal observer role and writable by users with the baremetal admin role. If the port is a member of a portgroup, the API will enforce that all ports in the portgroup have the same value in their physical network field.

Updates to the physical network field of ports will be restricted in the same way as for other connectivity related fields (link local connection, etc.) - they will be restricted to nodes in the `enroll`, `inspecting` and `manageable` states.

The API microversion will be bumped.

Client (CLI) impact

ironic CLI

The ironic CLI will not be updated.

openstack baremetal CLI

The openstack baremetal CLI will be updated to support getting and setting the `physical_network` field on ports.

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

None

Security impact

This change should increase the potential security level of an ironic bare metal cloud by supporting multiple segregated physical networks and honoring the physical network restrictions assigned by the operator.

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

In order to make use of this feature, deployers must tag ironic ports and portgroups with the physical network to which they are attached. This implies that they must have a mechanism to correctly determine this information.

Developer impact

None

Implementation

Assignee(s)

mgoddard

Work Items

- Modify the ironic port model to include a `physical_network` field.
- Modify the ironic ports REST API to support the `physical_network` field.
- Modify the openstack baremetal CLI to support the `physical_network` field.
- Modify the ironic `VIFPortIDMixin` plugin with the new port mapping algorithm.
- Modify the ironic `NeutronNetwork` network driver to be physical network- aware when creating neutron ports for cleaning and provisioning.
- Modify the ironic network drivers to add the physical network to neutron ports binding profiles.
- Add support for multiple (virtual) physical networks to DevStack.
- Update the ironic developer documentation to cover the use of physical networks.

Dependencies

None

Testing

Support will be added to DevStack for ironic environments with multiple (virtual) physical networks.

Upgrades and Backwards Compatibility

The proposed data model and algorithm changes are backwards compatible. A database migration will be provided to add the `physical_network` field to existing ports with a null value.

Documentation Impact

The ironic developer documentation will be updated to cover the use of this feature.

References

Add volume connection information for Ironic nodes

https://bugs.launchpad.net/ironic/+bug/1526231

This RFE introduces the changes in Ironic to support connecting and booting instances from remote volumes.

Problem description

When user starts bare metal instance with Cinder volume, Nova orchestrates the communication with Cinder and Ironic. The work flow of the boot process is as follows:

1. (Preparation) Administrator registers a node with initiator information.
2. User asks Cinder to create a boot volume.
3. User asks Nova to boot a node from the Cinder volume.
4. Nova calls Ironic to collect iSCSI/FC initiator information. Ironic collects initiator information and returns it to Nova.
5. Nova calls Cinder to attach the volume to the node. Cinder attaches the volume to the node and returns connection information which includes target information.
6. Nova passes the target information for the node to Ironic
7. Nova calls Ironic to spawn the instance. Ironic prepares the bare metal node to boot from the remote volume which is identified by target information and powers on the bare metal node.

In the work flow above, Nova calls Ironic to get/set initiator/target information (4 and 6) and also administrator calls Ironic to set initiator information (1) but currently Ironic has neither those information nor APIs for them.

Proposed change

- Add a new table named `volume_connectors` with the below fields:
  - id

        * Integer
        * PrimaryKeyConstraint
    - uuid
        * String(length=36)
        * UniqueConstraint
    - node_id
        * Integer
        * ForeignKeyConstraint(nodes.id)
    - created_at
        * DateTime
    - updated_at
        * DateTime
    - type (can have values iqn, ip, mac, wwnn, wwpn, net-id)
        * String(Length=32)
        * UniqueConstraint(type, connector_id)
    - connector_id
        * String(length=255)
        * UniqueConstraint(type, connector_id)
    - extra
        * Text

> **Note**
>
> Ironic should allow users to set IP and/or MAC address hardcorded into *connector_id* field because we cant put any assumptions on the storage network. It might be a Neutron network, or it might be a network that the OpenStack part of the deployment doesnt know about at all. It depends on deployment.

> **Note**
>
> *extra* field is text in the database but a dictionary (JSON-encoded dict) in the object

- Add a new table named `volume_targets` with the below fields:
    - id
        * Integer
        * PrimaryKeyConstraint
    - uuid

        * String(length=36)
        * UniqueConstraint
    - node_id
        * Integer
        * ForeignKeyConstraint(nodes.id)
    - created_at
        * DateTime
    - updated_at
        * DateTime
    - volume_type
        * String(length=64)
    - properties
        * Text
    - boot_index
        * Integer
        * UniqueConstraint(node_id, boot_index)
        * This is used for Ironic to distinguish the root volume. Similar to Nova, Ironic assumes volumes with boot index 0 are root device. (Nova associates a boot index with each block device and assumes volumes with boot index 0 are root volumes.)
    - volume_id
        * String(length=36)
    - extra
        * Text

Note

Ironic should clear the connected target information on node tear_down, just like it does for instance_info.

Note

properties and *extra* are text in the database but a dictionary (JSON-encoded dict) in the object

Note

The contents of the *properties* field depend on volume type. Reference information should be added in Bare Metal API document:

For iSCSI example:

```
{"auth_method": "CHAP",
 "auth_username": "XXX",
 "auth_password": "XXX",
 "target_iqn": "iqn.2010-10.com.example:vol-X",
 "target_portal": "192.168.0.123:3260",
 "volume_id": "12345678-...",
 "target_lun": 0,
 "access_mode": "rw",
 "target_discovered": false,
 "encrypted": false,
 "qos_specs": null}
```

For iSCSI multipath example:

```
{"auth_method": "CHAP",
 "auth_username": "XXX",
 "auth_password": "XXX",
 "target_iqns": ["iqn.2010-10.com.example:vol-X",
 "iqn.2010-10.com.example:vol-Y"],
 "target_portals": ["192.168.0.123:3260",
 "192.168.0.124:3260"],
 "volume_id": "12345678-...",
 "target_luns": [0, 1],
 "access_mode": "rw",
 "target_discovered": false,
 "encrypted": false,
 "qos_specs": null}
```

For fibre channel example:

```
{"device_path": "/dev/disk/by-path/pci-XXXX",
 "encrypted": false,
 "qos_specs": null,
 "target_lun": 1,
 "access_mode": "rw",
 "target_wwn": ["XXXX"]}
```

REST API masks credential information such as *auth_username* and *auth_password* in iSCSI and iSCSI multipath examples in order to avoid security risk.

- Add REST APIs end points to get/set values on them. For details see REST API Impact section.
  - /v1/volume/connectors
  - /v1/volume/targets
  - /v1/nodes/<node_uuid or name>/volume/connectors
  - /v1/nodes/<node_uuid or name>/volume/targets
- Add new capability flags in `node.properties['capabilities']`. These flags show whether or not the node can boot from volume with each backend. If it can boot from volume, we should set the flag to true.
  - iscsi_boot

  - fibre_channel_boot

> **Note**
>
> This should be set to true if the bare metal node supports booting from that specific volume. It might be populated manually by operator or by inspection, but that is not in the scope of this spec.

> **Note**
>
> In the future, Ironic will provide driver capabilities information. Nova can use that information to choose appropriate node.

- If a list of targets are specified, its up to the driver handling the deploy to take care of this. For multi-pathing, Ironic driver, bare metal hardware and the operating system should support it. If Ironic driver and bare metal hardware supports it, but instance operating system doesnt understand it, then it might lead to failure in booting the instance or corrupting the information in the Cinder volume.
- Information which is stored in volume_connector and volume_target tables are used in drivers in order to boot the node from volume. Changes for reference driver, driver interfaces are described in the spec[4].

Alternatives

- Saving connector information in a new node attribute like volume_initiator_info. This change has less impact on current code and API but proposed one has more benefits such as better integrity check, faster query from db and easier to store information related to a particular connector.
- Saving target information in a new node attribute like volume_target_info. This change has less impact on current code and API but proposed one has more benefits such as better integrity check, faster query from db and easier to store information related to a particular target.
- Saving target information in instance_info along with other instance related information. This seems to be straightforward because basically target volume information is related to the instance. In this case, `node.instance_info` is nested to store target information. This makes it difficult for users to manipulate target information, and for a driver to validate it. On the other hand, current approach can avoid nesting instance_info and so its easier to use those information. Note, ironic clears the target connection information on the node tear_down.
- Not implement storage of target and initiator information, which ultimately would not improve user experience and require manual post-deployment configuration for out-of-band control. For in-band use, Nova ironic driver can manage initiator information and it is proposed by jroll[2].

[4] https://review.opendev.org/#/c/294995
[2] https://review.opendev.org/#/c/184652/

Data model impact

- Add new type of object `VolumeConnector` in objects/volume_connector.py. It inherits IronicObject class. The new object will have the following fields:
  - `id`
  - `uuid`
  - `node_id`
  - `type`
  - `connector_id`
  - `extra`
  - `created_at` (defined in IronicObject class)
  - `updated_at` (defined in IronicObject class)
- Add new type of object `VolumeTarget` in objects/volume_target.py. It inherits IronicObject class. The new object will have the following fields:
  - `id`
  - `uuid`
  - `node_id`
  - `volume_type`
  - `properties`
  - `boot_index`
  - `volume_id`
  - `extra`
  - `created_at` (defined in IronicObject class)
  - `updated_at` (defined in IronicObject class)

State Machine Impact

None.

REST API impact

Six new REST API endpoints will be introduced with this change.

- `/v1/volume/connectors`
  - To set the volume connector (initiator) information:

    ```
    POST /v1/volume/connectors
    ```

    with the body containing the JSON description of the volume connector. It will return 201 on success, 400 if some required attributes are missing or having invalid value OR 409 if an entry already exists for the same volume connector.
  - To get information about all volume connectors:

```
GET /v1/volume/connectors
```

This operation will return a list of dictionaries. It contains information about all volume connectors:

```
{
    "volume_connectors":[
        {
            "connector_id": "<wwpn>",
            "links": [ ... ],
            "type": "wwpn",
            "uuid": "<uuid>",
        },
        {
            "connector_id": "<wwpn>",
            "links": [ ... ],
            ...
        },
        ...
    ]
}
```

This will return 200 on success

This operation can take parameters like `type`, `container_id`, `limit`, `marker`, `sort_dir`, and `fields`.

– To get detail information about all volume connectors:

```
GET /v1/volume/connectors/detail
```

The operation will return a list of dictionaries. It contains detailed information about all volume connectors:

```
{
    "volume_connectors":[
        {
            "connector_id": "<wwpn>",
            "created_at": "<created_date>",
            "extra": {},
            "links": [ ... ],
            "node_uuid": "<node_uuid>",
            "type": "wwpn",
            "updated_at": "<updated_date>",
            "uuid": "<uuid>",
        },
        {
            "connector_id": "<wwpn>",
            "created_at": "<created_date>",
            ...
        },
        ...
```

(continues on next page)

(continued from previous page)

```
    ]
}
```

It will return 200 on success.

This operation can take parameters like `type`, `container_id`, `limit`, `marker`, and `sort_dir`.

- It should be possible to pass `node` as a parameter which can be a node name or a node UUID to get all volume connectors for that particular node:

```
GET /v1/volume/connectors?node=<node_uuid or name>
GET /v1/volume/connectors/detail?node=<node_uuid or name>
```

It will return 200 on success or 404 if the node is not found.

- `/v1/volume/connectors/<volume_connector_uuid>`
  - To get detail information about a particular volume connector:

```
GET /v1/volume/connectors/<volume_connector_uuid>
```

This will return 200 on success or 404 if volume connector is not found.

  - To update a particular volume connector:

```
PATCH /v1/volume/connectors/<volume_connector_uuid>
```

This will return 200 and the representation of the updated resource on success and 404 if volume connector is not found.

Note

Updating connector information when the node is in POWER_ON or REBOOT state is blocked. It means that users need to make sure the node is in POWER_OFF state before updating connector information. When connector information is updated, driver should update node configuration.

  - To delete volume connector:

```
DELETE /v1/volume/connectors/<volume_connector_uuid>
```

It will return 204 on success or 404 if volume connector is not found or 400 if the node is not in POWER_OFF state.

- `/v1/nodes/<node_uuid or name>/volume/connectors`
  - To get all the volume connectors information for a node:

```
GET ``/v1/nodes/<node_uuid or name>/volume/connectors``
```

- `/v1/volume/targets`
  - To set the volume target information:

```
POST /v1/volume/targets
```

with the body containing the JSON description of the volume target. It will return 201 on success, 400 if some required attributes are missing or having invalid value OR 409 if an entry already exists for the same volume target.

– To get information about all volume targets:

```
GET /v1/volume/targets
```

This operation will return a list of dictionaries. It contains information about all volume targets:

```
{
    "volume_targets":[
        {
            "boot_index", "<boot_index>",
            "links": [ ... ],
            "uuid": "<uuid>",
            "volume_id": "<volume_id>"
            "volume_type": "<volume_target_type>",
        },
        {
            "boot_index", "<boot_index>",
            "links": [ ... ],
            ...
        },
        ...
    ]
}
```

This will return 200 on success.

This operation can take parameters like `boot_index`, `volume_id`, `volume_type`, `limit`, `marker`, `sort_dir`, and `fields`.

– To get details information about all volume targets:

```
GET /v1/volume/targets/detail
```

The operation will return a list of dictionaries. It contains detailed information about all volume targets:

```
{
    "volume_targets":[
        {
            "boot_index": "<boot_index>",
            "created_at": "<created_date>",
            "extra": {},
            "links": [ ... ],
            "node_uuid": "<node_uuid>",
            "properties" : { "<target_information>" },
```

(continues on next page)

(continued from previous page)

```
            "updated_at": "<updated_date>",
            "uuid": "<uuid>",
            "volume_id": "<volume_id>",
            "volume_type": "<volume_target_type>",
        },
        {
            "boot_index": "<boot_index>",
            "created_at": "<created_date>",
            ...
        },
        ...
    ]
}
```

It will return 200 on success.

This operation can take parameters like `boot_index`, `volume_id`, `volume_type`, `limit`, `marker`, and `sort_dir`.

Note

properties may include credential information. This API will mask it to avoid security risk.

- It should be possible to pass `node` as a parameter which can be a node name or a node UUID to get all volume targets for that particular node:

```
GET /v1/volume/targets?node=<node_uuid or name>
GET /v1/volume/targets/detail?node=<node_uuid or name>
```

It will return 200 on success or 404 if the node is not found.

- `/v1/volume/targets/<volume_target_uuid>`
  - To get detailed information about a particular volume target:

```
GET /v1/volume/targets/<volume_target_uuid>
```

This will return 200 on success or 404 if volume target is not found.

  - To update a particular volume target:

```
PATCH /v1/volume/targets/<volume_target_uuid>
```

This will return 200 and the representation of the updated resource on success, 404 if volume target is not found or 400 if the node is not POWER_OFF state.

Note

Updating target information when the node is in POWER_ON or REBOOT state is blocked. It means that users need to make sure the node is in POWER_OFF state before updating target

information. When target information is updated, driver should update node configuration.

  - To delete volume target:

    ```
    DELETE /v1/volume/targets/<volume_target_uuid>
    ```

    It will return 204 on success, 404 if volume target is not found or 400 if the node is not in POWER_OFF state.

- `/v1/nodes/<node_uuid or name>/volume/targets`
  - To get all the volume targets information for a node:

    ```
    GET ``/v1/nodes/<node_uuid or name>/volume/targets``
    ```

- `/v1/nodes/<node_uuid or name>/volume/targets`
  - To get all the volume targets information for a node:

    ```
    GET ``/v1/nodes/<node_uuid or name>/volume/targets``
    ```

The endpoint `GET /v1/nodes/detail` will provide the volume connectors and targets information for the node with links to them. Also, the endpoint `GET /v1/nodes/<node_uuid or name>` will provide the volume connectors and targets information for the specified node.

For the above REST API changes, micro version will be bumped and 406 will be raised if newer endpoints are accessed with a lesser micro version.

Client (CLI) impact

- A new `VolumeConnectorManager` will be added to `ironicclient` to get/set connector information for the node. Also the CLI will be modified as follows:

```
ironic volume-connector-create --node <node> --type <type>
                               --connector_id <connector_id>
                               [-e <key=value>] [-u <uuid>]
ironic volume-connector-delete <uuid> [<uuid>]
ironic volume-connector-list [--detail] [--type <type>]
                             [--connector_id <connector_id>]
                             [--limit <limit>] [--marker <uuid>]
                             [--sort-key <field>] [--sort-dir <direction>]
                             [--fields <field> [<field> ...]]
ironic volume-connector-show [--fields <field> [<field> ...]] <uuid>
ironic volume-connector-update <uuid> <op> <path=value> [<path=value> ...]

ironic node-volume-connector-list [--detail] [--limit <limit>]
                                  [--marker <uuid>] [--sort-key <field>]
                                  [--sort-dir <direction>]
                                  [--fields <field> [<field> ...]]
                                  <node>
```

- A new `VolumeTargetManager` will be added to `ironicclient` to get/set target information for the node. Also the CLI will be modified as follows:

```
ironic volume-target-create --node <node> --volume_type <volume_type>
                            --volume_id <volume_id>
                            [--properties <key=value>]
                            [--boot_index <boot_index>]
                            [-e <key=value>] [-u <uuid>]
ironic volume-target-delete <uuid> [<uuid>]
ironic volume-target-list [--detail] [--volume_type <volume_type>]
                          [--volume_id <volume_id>]
                          [--boot_index <boot_index>] [--limit <limit>]
                          [--marker <uuid>] [--sort-key <field>]
                          [--sort-dir <direction>]
                          [--fields <field> [<field> ...]]
ironic volume-target-show [--fields <field> [<field> ...]] <uuid>
ironic volume-target-update <uuid> <op> <path=value> [<path=value> ...]

ironic node-volume-target-list [--detail] [--limit <limit>]
                               [--marker <uuid>] [--sort-key <field>]
                               [--sort-dir <direction>]
                               <node>
```

- New objects, `CreateBaremetalVolumeConnector`, `DeleteBaremetalVolumeConnector`, `ListBaremetalVolumeConnector`, `SetBaremetalVolumeConnector`, `ShowBaremetalVolumeConnector`, and `UnsetBaremetalVolumeConnector` will be added to `openstackclient` plugin to get/set connector information for the node. Also the CLI will be modified as follows:

```
openstack baremetal volume connector create [-h]
                                            [-f {json,shell,table,value,
↪yaml}]
                                            [-c COLUMN]
                                            [--max-width <integer>]
                                            [--noindent] [--prefix PREFIX]
                                            --node <node_uuid> --type
↪<type>
                                            --connector_id <connector_id>
                                            [--extra <key=value>]
                                            [--uuid <uuid>]
openstack baremetal volume connector delete [-h] <connector> [<connector>]
openstack baremetal volume connector list [-h]
                                          [-f {json,shell,table,value,
↪yaml}]
                                          [-c COLUMN]
                                          [--max-width <integer>]
                                          [--noindent]
                                          [--quote {all,minimal,none,
↪nonnumeric}]
                                          [--limit <limit>]
                                          [--marker <uuid>]
                                          [--sort <key>[:<direction>]]
                                          [--long | fields <field [field]␣
```

(continues on next page)

(continued from previous page)

```
↪...>]
openstack baremetal volume connector set [-h] [--node <node>]
                                         [--type <type>]
                                         [--connector_id <connector_id>]
                                         [--extra <key=value>] <connector>
openstack baremetal volume connector show [-h]
                                          [-f {json,shell,table,value,
↪yaml}]
                                          [-c COLUMN]
                                          [--max-width <integer>]
                                          [--noindent] [--prefix PREFIX]
                                          [--fields <field> [<field> ...]]
                                          <connector>
openstack baremetal volume connector unset [-h] [--extra <key>]
↪<connector>
```

- New objects, `CreateBaremetalVolumeTarget`, `DeleteBaremetalVolumeTarget`, `ListBaremetalVolumeTarget`, `SetBaremetalVolumeTarget`, `ShowBaremetalVolumeTarget`, and `UnsetBaremetalVolumeTarget` will be added to `openstackclient` plugin to get/set target information for the node. Also the CLI will be modified as follows:

```
openstack baremetal volume target create [-h]
                                         [-f {json,shell,table,value,yaml}
↪]
                                         [-c COLUMN] [--max-width
↪<integer>]
                                         [--noindent] [--prefix PREFIX]
                                         --node <node_uuid> --type <type>
                                         --volume_id <volume_id>
                                         [--properties <key=value>]
                                         [--boot_index <boot_index>]
                                         [--extra <key=value>]
                                         [--uuid <uuid>]
openstack baremetal volume target delete [-h] <target> [<target>]
openstack baremetal volume target list [-h]
                                       [-f {json,shell,table,value,yaml}]
                                       [-c COLUMN] [--max-width <integer>]
                                       [--noindent]
                                       [--quote {all,minimal,none,
↪nonnumeric}]
                                       [--limit <limit>] [--marker <uuid>]
                                       [--sort <key>[:<direction>]]
                                       [--long | fields <field [field] ...
↪>]
openstack baremetal volume target set [-h] [--node <node>] [--type <type>]
                                      [--volume_id <volume_id>]
                                      [--properties <key=value>]
                                      [--boot_index <boot_index>]
                                      [--extra <key=value>] <target>
```

(continues on next page)

(continued from previous page)

```
openstack baremetal volume target show [-h]
                                       [-f {json,shell,table,value,yaml}]
                                       [-c COLUMN] [--max-width <integer>]
                                       [--noindent] [--prefix PREFIX]
                                       [--fields <field> [<field> ...]]
                                       <target>
openstack baremetal volume target unset [-h]
                                        [--properties <key>]
                                        [--boot_index] [--extra <key>]
                                        <target>
```

RPC API impact

Four new rpcapi method `update_volume_connector`, `destroy_volume_connector`, `update_volume_target`, and `destroy_volume_target` will be added.

- `update_volume_connector`

  This method takes context and volume connector object as input and returns updated volume connector object.

- `destroy_volume_connector`

  This method takes context and volume connector object as input.

- `update_volume_target`

  This method takes context and volume target object as input and returns updated volume target object.

- `destroy_volume_target`

  This method takes context and volume target object as input.

Driver API impact

None.

Nova driver impact

When spawning a new instance, Nova Ironic virt driver queries Ironic (through API) to find out the volume connector information. It passes the volume connector information to Cinder which returns the target information. This is then passed down to Ironic. Detailed information about Nova Ironic driver can be found in the spec[5].

Ramdisk impact

None

[5] https://review.opendev.org/#/c/211101/

Security impact

None.

Note

As for FC zoning, Cinder takes care of it[6].

Other end user impact

None.

Scalability impact

None.

Performance Impact

This may extend the time required for nova boot/delete, but its not a big impact and its important for enterprise users.

Other deployer impact

- If administrators want to provide boot from volume feature, they need to fill out following initiator information before activating the node.
  - iSCSI:
    - ip
    - iqn
    - mac

    Note

    ip may be omitted when Neutron is used to manage the storage network.

  - FC:
    - wwnn
    - wwpn

  Administrators need to set the node.properties[capabilities] (iscsi_boot and/or fibre_channel_boot) true.

  Its better if inspection automatically collects and registers them. For example, in the case of a node with FC-HBA, inspection(in-band) can get wwnn and wwpn from sysfs like following:

```
# cat /sys/class/scsi_host/host*/device/fc_host/host*/node_name
# cat /sys/class/scsi_host/host*/device/fc_host/host*/port_name
```

[6] http://docs.openstack.org/mitaka/config-reference/block-storage/fc-zoning.html

- If users want to boot a node from volume in Ironic standalone mode, they need additional tooling to leverage this functionality. For example, that tool needs to do something like:
  - Get initiator information from Ironic
  - Call the storage management tool with initiator information to create a new volume (maybe from template) and attach it to the initiator
  - Get target information from storage management tool
  - Put target information into Ironic

Developer impact

Driver developers can consume the information mentioned above to write boot from volume support in their driver. The details about reference driver and driver interface specs are described in [4].

Implementation

Assignee(s)

Primary assignee:
satoru-moriya-br

Other contributors:
rameshg87

Work Items

- Create new table named volume_connectors and volume_targets
- Create new DB API methods
- Create new Object named VolumeConnector and VolumeTarget
- Create new RPC API methods
- Create new REST API endpoints
- Document the changes
- Enhance inspector to register connector information if available
- Enhance Client(CLI) to get/set connector and target information
- Enhance Nova-Ironic driver to support boot from volume with these APIs

Dependencies

None

Testing

- Unit tests will be added/updated to cover the changes.
- Tempest tests will be added to Ironic to ensure that the following newly added API endpoints work correctly.

Upgrades and Backwards Compatibility

Add a migration script for database.

Documentation Impact

Documentations such as Installation guide and api-ref will be updated to explain the newly added fields and end points.

- Installation guide:

  http://docs.openstack.org/developer/ironic/deploy/install-guide.html

- api-ref documentation:

  http://developer.openstack.org/api-ref/baremetal/index.html

References

5.9.3 8.0

Redfish hardware type and interfaces

https://bugs.launchpad.net/ironic/+bug/1526477

This specification proposes the addition of a new `redfish` hardware type, power and management interfaces in order to support ironic deployment on Redfish compliant servers.

Problem description

The Distributed Management Task Force (DMTF) has published a new specification called Redfish (refer to http://www.dmtf.org/standards/redfish) to provide a RESTful based API to manage servers in a standard way. This specification aims at adding support to ironic for controlling Redfish compliant servers.

Proposed change

This spec proposes adding a new `redfish` hardware type, power and management interfaces. None of which will be enabled by default.

The new interfaces will use the sushy library in order to handle the communication between the driver and the Redfish controller.

This library may be switched to python-redfish in the future after re-evaluation by the ironic community. The switch to python-redfish is outside the scope of this specification, but it should not cause any public interface to be changed. It would most likely involve code changes and new configuration options.

Note that no OEM specific extension will be supported.

Alternatives

None

Data model impact

None

RPC API impact

None

State Machine Impact

None

REST API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

None

Security impact

None

Client (CLI) impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

The following driver_info fields are required while enrolling nodes into ironic:

- `redfish_address`: The URL address to the Redfish controller. It should include scheme and authority portion of the URL. For example: https://mgmt.vendor.com
- `redfish_system_id`: The canonical path to the ComputerSystem resource that the driver will interact with. It should include the root service, version and the unique resource path to a ComputerSystem within the same authority as the redfish_address property. For example: /redfish/v1/Systems/1
- `redfish_username`: User account with admin/server-profile access privilege. Although this property is not mandatory its highly recommended to set a username. Optional
- `redfish_password`: User account password. Although this property is not mandatory its highly recommended to set a password. Optional
- `redfish_verify_ca`: This property contains either a boolean value, a path to a CA_BUNDLE file or directory with certificates of trusted CAs. If set to True the driver will verify the host certificates; if False the driver will ignore verifying the SSL certificate; If its a path the driver will use the specified certificate or one of the certificates in the directory. Defaults to True. Optional

For more information about the expected syntax of the `redfish_system_id` property check the Resource identifier property section of the DMTF specification.

The following new configuration variables are proposed (and their default values) to be added to the conductor variable group:

- [redfish]/`connection_attempts`

  Maximum number of attempts to try to connect to Redfish.

- [redfish]/`connection_retry_interval`

  Number of seconds to wait between attempts to connect to Redfish.

Developer impact

None

Implementation

Assignee(s)

Primary assignee:

- bcornec
- lucasagomes

Other contributors:

- ribaudr

Work Items

- Add a new `redfish` hardware type, power and management interfaces.
- Write unit-tests for the new code.
- Modify the ironic DevStack module to setup a virtual environment that is able to test nodes using the new Redfish driver.
- Write documentation.

Dependencies

The new `redfish` power and management interfaces will require the sushy library to be installed on the conductor node.

Testing

Unit-tests will be implemented for Redfish support.

DevStack will be updated to setup the nodes with the redfish driver and the libvirt mockup that is shipped with Sushy allowing it to be tests in gate against virtual machines.

Upgrades and Backwards Compatibility

This driver will not break any compatibility with either the REST API or the RPC API.

Documentation Impact

- Updating ironic documentation section Enabling Drivers with Redfish related instructions.
- Updating ironic install-guide documentation section Setup the drivers for the Bare Metal service.

References

- Redfish DMTF: http://www.dmtf.org/standards/redfish
- Sushy library: https://github.com/openstack/sushy
- python-redfish library: https://github.com/openstack/python-redfish

OSC commands to get descriptions of driver-related information

- https://bugs.launchpad.net/python-ironicclient/+bug/1619052
- https://bugs.launchpad.net/python-ironicclient/+bug/1619053

Almost all the ironic CLI commands have corresponding OpenStack Client (OSC) commands. This was done in Implementation of ironic commands as an OSC plugin.

There are two ironic CLI commands:

- `ironic driver-properties` and
- `ironic driver-raid-logical-disk-properties`

that do not have corresponding OSC commands. This specification proposes OSC commands for them.

Properties of hardware types with non-default interfaces is addressed in RFE to get properties for a dynamic driver and non-default interfaces. That proposal will most likely result in a richer REST API (and corresponding OSC commands). In light of that, we propose simple OSC commands with the goal of providing equivalent behaviours to the ironic CLI commands, and nothing more.

Problem description

This table shows the ironic driver-related commands and their associated OSC commands.

ironic	openstack baremetal
driver-get-vendor-passthru-methods <driver>	driver passthru list <driver>
driver-list	driver list
driver-show <driver>	driver show <driver>
driver-vendor-passthru <driver> <method>	driver passthru call <driver> <method>
driver-properties <driver>	?
driver-raid-logical-disk-properties <driver>	?

The question is what to use for the OSC commands corresponding to these ironic commands:

- `ironic driver-properties <driver>`
- `ironic driver-raid-logical-disk-properties <driver>`.

These commands return tables with two columns; the names and descriptions for driver-related information.

To complicate things a bit, there is a lack of symmetry. Although the names (and descriptions) are available via the driver, the values for these are specified via the nodes that use these drivers, rather than directly via the driver.

There have been a few discussions related to this:

- http://lists.openstack.org/pipermail/openstack-dev/2016-September/103490.html
- briefly discussed at the OpenStack PTG in Atlanta in February 2017, see How to call (a name for it) the OSC command for listing RAID properties? (dtantsur) in the PTG operations etherpad.

Properties of hardware types with non-default interfaces is addressed in RFE to get properties for a dynamic driver and non-default interfaces. That proposal will most likely result in a richer REST API (and corresponding OSC commands). In light of that, we propose simple OSC commands with the goal of providing equivalent behaviours to the ironic CLI commands, and nothing more.

ironic driver-properties <driver>

The `ironic driver-properties <driver>` command returns the properties of a driver that must be specified via the nodes that use that driver.

For example, `ironic driver-properties agent_ipmitool` returns a table of information:

```
+--------------------------+-----------------------------------------------+
| Property                 | Description                                   |
+--------------------------+-----------------------------------------------+
| deploy_forces_oob_reboot | Whether Ironic should force a reboot of t...|
| deploy_kernel            | UUID (from Glance) of the deployment kern...|
```

(continues on next page)

(continued from previous page)

```
| deploy_ramdisk           | UUID (from Glance) of the ramdisk that is...|
| image_http_proxy         | URL of a proxy server for HTTP connection...|
| image_https_proxy        | URL of a proxy server for HTTPS connectio...|
| image_no_proxy           | A comma-separated list of host names, IP ...|
| ...                      | ...                                         |
+--------------------------+---------------------------------------------+
```

The values for these driver properties are set per node, for each node that uses the driver. The information is in the nodes `driver_info` dictionary field. They can be set when creating a node or when updating a node, for example with `openstack baremetal node set <node> --driver-info`.

ironic driver-raid-logical-disk-properties <driver>

The `ironic driver-raid-logical-disk-properties <driver>` command returns the RAID logical disk properties that can be specified for a particular driver.

For example, `ironic driver-raid-logical-disk-properties agent_ipmitool` returns a table of information:

```
+--------------------------+---------------------------------------------+
| Property                 | Description                                 |
+--------------------------+---------------------------------------------+
| controller               | Controller to use for this logical disk. ...|
| disk_type                | The type of disk preferred. Valid values ...|
| interface_type           | The interface type of disk. Valid values ...|
| is_root_volume           | Specifies whether this disk is a root vol...|
| number_of_physical_disks | Number of physical disks to use for this ...|
| physical_disks           | The physical disks to use for this logica...|
| raid_level               | RAID level for the logical disk. Valid va...|
| share_physical_disks     | Specifies whether other logical disks can...|
| size_gb                  | Size in GiB (Integer) for the logical dis...|
| ...                      | ...                                         |
+--------------------------+---------------------------------------------+
```

The values for these disk properties are set per node, for each node that uses the driver and RAID. This information is in the nodes `target_raid_config` field and can be set (in JSON format) via the nodes set raid config API, or via `openstack baremetal node set <node> --target-raid-config`.

Proposed change

The OpenStackClient command structure is:

```
openstack [<global-options>] <object-1> <action> [<object-2>] [<command-
↪arguments>]
```

ironic driver-properties <driver>

The <object-1> part will be baremetal driver property.

For the <action>, there were discussions at the PTG about the merits of using list versus show. In the context of existing OSC commands using these two action words, neither of these action words seems

to fit with the two commands, since they return descriptions (or documentation) on what is available. However, since this will very likely be replaced by a richer set of OSC commands (resulting from RFE to get properties for a dynamic driver and non-default interfaces), we keep it simple and use list. (Using list might imply that a corresponding show exists to drill down and get more information. The converse is true too though; using show might imply a corresponding list command.)

The OSC command is:

```
openstack baremetal driver property list <driver>
```

For example:

```
$ openstack baremetal driver property list agent_ipmitool

+--------------------------+---------------------------------------------+
| Property                 | Description                                 |
+--------------------------+---------------------------------------------+
| deploy_forces_oob_reboot | Whether Ironic should force a reboot of t...|
| deploy_kernel            | UUID (from Glance) of the deployment kern...|
| deploy_ramdisk           | UUID (from Glance) of the ramdisk that is...|
| image_http_proxy         | URL of a proxy server for HTTP connection...|
| image_https_proxy        | URL of a proxy server for HTTPS connectio...|
| image_no_proxy           | A comma-separated list of host names, IP ...|
| ...                      | ...                                         |
+--------------------------+---------------------------------------------+
```

ironic driver-raid-logical-disk-properties <driver>

Again, since this will very likely be replaced by a richer set of OSC commands (that will result from RFE to get properties for a dynamic driver and non-default interfaces), we propose a simple OSC command.

<object-1> would be baremetal driver raid property and <action> would be list:

```
openstack baremetal driver raid property list <driver>
```

Alternatives

There are alternatives, but in the interest of keeping this simple and deprecating it when RFE to get properties for a dynamic driver and non-default interfaces is available, we are focussed here on only providing OSC-equivalent commands that may not be flexible or extensible, but provide equivalence to the ironic CLI ones.

Data model impact

None.

State Machine Impact

None.

REST API impact

None.

Client (CLI) impact

This specification is about the OSC CLI.

ironic CLI

None.

openstack baremetal CLI

See above.

RPC API impact

None.

Driver API impact

None.

Nova driver impact

None.

Ramdisk impact

None.

Security impact

None.

Other end user impact

None.

Scalability impact

None.

Performance Impact

None.

Other deployer impact

None.

Developer impact

None.

Implementation

Assignee(s)

Primary assignee: rloo (Ruby Loo)

Other contributors: galyna (Galyna Zholtkevych)

Work Items

- Code the two OSC commands in python-ironicclient.

Dependencies

None.

Testing

Unit and functional testing similar to what exists for other OSC commands.

Upgrades and Backwards Compatibility

None.

Documentation Impact

None. ironic doesnt have OSC-command related documentation.

References

- Implementation of ironic commands as an OSC plugin
- OpenStackClient command
- PTG operations etherpad
- updating a node
- nodes set raid config
- http://lists.openstack.org/pipermail/openstack-dev/2016-September/103490.html
- RFE to get properties for a dynamic driver and non-default interfaces

5.10 Ocata

5.10.1 7.0

Add notifications about resources CRUD and node states

https://bugs.launchpad.net/ironic/+bug/1606520

This spec proposes addition of new notifications to ironic: CRUD (create, update, or delete) of resources and node state changes for provision state, maintenance and console state.

Problem description

Resource indexation services like Searchlight[1] require notifications about creation, update or deletion of a resource. Currently CRUD notifications are not implemented in ironic. Creating an efficient plugin for Searchlight is impossible without these notifications. Ironic node notifications for provision state, maintenance and console state also could be used by Searchlight plugin in order to keep Searchlights index of ironic resources up-to-date.

Apart from searchlight, there is a use case of monitoring service, that caches all notification payloads along with event type, like start/end/error/<etc> and an operator can query this service to see if ironic is behaving properly. For example, if there are much more start notifications for node create, than there are end notifications, it may mean that the database is not behaving properly, or messaging is having a hard time delivering messages between API and conductor. That is a separate case from searchlight: searchlight for example does not need to know the payload of the node create start notification, as there is no actual node yet, but for monitoring purposes, it may be useful.

Proposed change

As a general note for all CRUD notifications, `*.start` and `*.error` event payloads will be ignored by Searchlight, as in both cases it would mean that resource representation has not changed, or in case of `*create*` notifications, that the resource was not created.

Node CRUD notifications

The following event types will be added:

- baremetal.node.create.start;
- baremetal.node.create.end;
- baremetal.node.create.error;
- baremetal.node.update.start;
- baremetal.node.update.end;
- baremetal.node.update.error;
- baremetal.node.delete.start;
- baremetal.node.delete.end;
- baremetal.node.delete.error.

Priority level - INFO or ERROR (for error status). Payload contains all fields from base `NodePayload` with additional fields: `chassis_uuid`, `instance_info`, `driver_info`. Secrets in the node fields will

[1] https://wiki.openstack.org/wiki/Searchlight

be masked. `raid_config` and `target_raid_config` fields are excluded because they can contain low-level disk and vendor information. If/when there is a use case for them, they can be added in the future. All these notifications will be implemented at the API level.

Port CRUD notifications

The following event types will be added:

- baremetal.port.create.start;
- baremetal.port.create.end;
- baremetal.port.create.error;
- baremetal.port.update.start;
- baremetal.port.update.end;
- baremetal.port.update.error;
- baremetal.port.delete.start;
- baremetal.port.delete.end;
- baremetal.port.delete.error.

Priority level - INFO or ERROR (for error status). Payload contains these fields: `uuid`, `node_uuid`, `address`, `extra`, `local_link_connection`, `pxe_enabled`, `created_at`, `updated_at`. These notifications will be implemented at the API level. In addition, baremetal.port.create.* will be emitted by the ironic-conductor service when driver creates a port (examples are[2] and[3]).

Chassis CRUD notifications

The following event types will be added:

- baremetal.chassis.create.start;
- baremetal.chassis.create.end;
- baremetal.chassis.create.error;
- baremetal.chassis.update.start;
- baremetal.chassis.update.end;
- baremetal.chassis.update.error;
- baremetal.chassis.delete.start.
- baremetal.chassis.delete.end.
- baremetal.chassis.delete.error;

Priority level - INFO or ERROR (for error status). Payload contains these fields: `uuid`, `extra`, `description`, `created_at`, `updated_at`. All these notifications will be implemented at the API level.

[2] https://github.com/openstack/ironic/blob/2c76da5f437c5fc2f4022e8705e74fed0a46bebb/ironic/drivers/modules/irmc/inspect.py#L177

[3] https://github.com/openstack/ironic/blob/2c76da5f437c5fc2f4022e8705e74fed0a46bebb/ironic/drivers/modules/ilo/inspect.py#L56

Node provision state notifications

Will be implemented via TaskManager methods (and emitted by the ironic-conductor service).

Types of events for node provision state:

- baremetal.node.provision_set.start;
- baremetal.node.provision_set.end;
- baremetal.node.provision_set.error;
- baremetal.node.provision_set.success.

Types of state changing in ironic and corresponding events:

- Start transition, spawning a working thread: start notification with INFO level.
- End transition, cleaning `target_provision_state`: end notification with INFO level.
- Error events processing: error notification with ERROR level.
- Change `provision_state` without starting a worker that is not end or error: success notification with INFO level. Examples are DEPLOYING <-> DEPLOYWAIT, AVAILABLE -> MANAGEABLE.

Payload contains all fields from base `NodePayload` with additional fields: `instance_info`, `previous_provision_state`, `previous_target_provision_state`, `event` (FSM event that triggered the state change). To efficiently use the provision state notifications all related node changes (like setting of `last_error`, `maintenance`) should be done before event processing.

Node maintenance notifications

The following event types will be added:

- baremetal.node.maintenance_set.start;
- baremetal.node.maintenance_set.end;
- baremetal.node.maintenance_set.error.

Priority level - INFO or ERROR (for error status). Payload contains all fields from base `NodePayload`. All these notifications will be implemented at the API level and reflect maintenance changes to a node due to a user request. There wont be any explicit node maintenance notifications for maintenance changes done internally by ironic. Since these internal changes occur as a result of trying to change the nodes state (e.g. provision, power), one of the other notifications that is emitted will cover these internal maintenance changes.

Node console notifications

The following event types will be added:

- baremetal.node.console_set.start;
- baremetal.node.console_set.end;
- baremetal.node.console_set.error;
- baremetal.node.console_restore.start;
- baremetal.node.console_restore.end;

- baremetal.node.console_restore.error.

`console_set` action is used when start or stop console is initiated via API request, `console_restore` action is used when `console_enabled` flag is already enabled in the DB for node and console restart via driver is required (due to dead or restarted ironic-conductor process). Priority level - INFO or ERROR (for error status). Payload contains all fields from base `NodePayload`. All these notifications will be implemented in the ironic-conductor, because setting of a nodes console is an asynchronous request, so ironic-conductor can easily emit notifications for the start/end of the change.

Alternatives

Periodically polling ironic resources via API.

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

None

Security impact

None

Other end user impact

None

Scalability impact

If notifications are enabled, they can create high load on the message bus during node deployments on large environments.

Performance Impact

None

Other deployer impact

Deployers should set already existing `notification_level` config options properly.

Developer impact

- If developer creates resources in the driver, proper notification should be emitted.
- For provision state change all related node updates should be done before event processing.

Implementation

Assignee(s)

Primary assignee:

- yuriyz

Other contributors:

- vdrok
- mariojv

Work Items

- Implement node provision state change notifications.
- Implement CRUD notifications and node maintenance notifications.
- Implement console notifications.
- Add notifications to the current ironic code that creates resources in the drivers.
- Fix ironic code with node updates after event processing.

Dependencies

Patch with base `NodePayload`[4].

[4] https://review.opendev.org/#/c/321865/

Testing

Unit tests will be added.

Upgrades and Backwards Compatibility

None

Documentation Impact

New notifications feature will be documented.

References

Driver composition reform

https://bugs.launchpad.net/ironic/+bug/1524745

This spec suggests revamping how we name and compose drivers from interfaces. It will allow having one vendor driver with options configurable per node instead of many drivers for every vendor.

Problem description

Our driver interface matrix has become increasingly complex. To top it all off nowadays we have many interfaces that can be used for every driver. To name a few:

- `boot`: most drivers support PXE and iPXE, while some also support virtual media; support for *petitboot* bootloader is proposed.
- `deploy`: two deploy approaches are supported: write image via iSCSI or write it directly from within the agent.
- `inspect`: there is generic inspection using ironic-inspector, but some drivers also allow out-of-band inspection. This feature is optional, so we should provide a way to disable it.

Currently weve ended up with a complex and really confusing naming scheme. For example:

- `pxe_ipmitool` uses PXE or iPXE for boot and iSCSI for deploy.
- `agent_ipmitool` actually also uses PXE or iPXE, but it does not use iSCSI.
- To top it all, `pxe_ipmitool` is actually using agent!
- To reflect all the possibilities, the names would have to be more like `pxe_iscsi_ipmitool`, `ipxe_iscsi_ipmitool`, `pxe_direct_ipmitool`, `ipxe_direct_ipmitool`, etc.
- Now repeat the same with every power driver we have.

Proposed change

Introduction

The following concepts are used in this spec:

vendor
: The driving force behind a specific driver. It can be hardware vendors or the ironic team itself in case of generic drivers, such as IPMI. This also includes out-of-tree drivers.

hardware interface (or just interface)
A notion replacing the term driver interface - a set of functionality dealing with some aspect of bare metal provisioning in a vendor-specific way. For example, right now we have `power`, `deploy`, `inspect`, `boot` and a few more interfaces.

hardware type
A family of hardware supporting the same set of interfaces from the ironic standpoint. This can be as wide as all hardware supporting the IPMI protocol or as narrow as several hardware models supporting some specific interfaces.

driver
A thin object containing links to hardware interfaces. Before this spec *driver* meant roughly the same as *hardware type* means in this spec.

classic driver
An ironic driver from before this spec: a class with links to interfaces hardcoded in the Python code.

dynamic driver
A *driver* created at run time with links to interfaces generated based on information in the node record (including hardware type and interfaces).

With this spec we are going to achieve the following goals:

- Make *vendors* in charge of defining a set of supported interface implementations in priority order.
- Allow *vendors* to guarantee that unsupported interface implementations will not be used with hardware types they define. This is done by having a hardware type list all interfaces it supports.
- Allow 3rd parties to create out-of-tree *hardware types* that allow them to maximize their reuse of the in-tree interface implementations.
- Make *hardware type* definition as declarative as possible.
- Allow a user to switch *hardware type* for a node, just as it was possible to change a driver before this spec.
- Allow a user to switch between *interface* implementations supported by a *hardware type* for a node via the bare metal API.

Configuration

- A *hardware type* is defined as a Python class - see *Hardware Types* for details. An entry point is created to provide a simple name for each *hardware type*, for example:

```
ironic.hardware.types =
    generic-ipmi = ironic.hardware.ipmi:GenericIpmiHardware
    ilo-gen8 = ironic.hardware.ilo:iLOGen8Hardware
    ilo-gen9 = ironic.hardware.ilo:iLOGen9Hardware
```

- The list of *hardware interfaces* is still hardcoded in the Python code and cannot be extended by plugins. The interfaces are implemented in the same way as before this spec: by subclassing an appropriate abstract class from ironic.drivers.base.
- For each *hardware interface*, all implementations get their own entrypoint and a unique name, for example:

```
ironic.hardware.interfaces.power =
    ipmitool = ironic.drivers.modules.ipmitool:IpmitoolPower
```

- Compatibility between *hardware types* and *hardware interface* implementations is expressed in the Python code - see *Hardware Types* for details.
- Create a new configuration option `enabled_hardware_types` with a list of enabled *hardware types*. This will not include *classic drivers* which are enabled by the existing `enabled_drivers` option.
- Create a family of configuration options `default_<INTERFACE>_interface` that allows an operator to explicitly set a default interface for new nodes upon creation, if one is not specified in the creation request.

  Here `<INTERFACE>` is a type of interface: power, management, etc.
- Create a family of configuration options `enabled_<INTERFACE>_interfaces` with a list of enabled implementations of each *hardware interface* that are available for use in the ironic deployment.

  The default value, if provided by the `default_<INTERFACE>_interface`, must be in this list, otherwise a conductor will fail to start.
- If no interface implementation is explicitly requested by a user in a node creation request, use the value calculated as follows:
  - If `default_<INTERFACE>_interface` is set, use its value. Return an error if it is not supported by the *hardware type* of the node.
  - Otherwise choose the first available interface implementation from an intersection of the `enabled_<INTERFACE>_interfaces` as defined in the deployments configuration and the *hardware_type*s priority ordered list of supported_<INTERFACE>_interfaces. Return an error, if this intersection is empty.

  This calculated default will be stored in the database entry for the node upon creation.
- Change how we load drivers instead of one singleton instance of a driver, well have an instance of *dynamic driver* per node, containing links to hardware interface implementations (just like today).

  However, interface implementations themselves will stay singletons, and will be preloaded during the start up and stored in the conductor.

  Conductor will fail to start if any **enabled** *hardware types* or *interface* implementations cannot be loaded (e.g. due to missing dependencies).

  Note

  While its technically possible to enable interfaces that are not used in any of enabled *hardware types*, they will not get loaded in this case.

  The *classic drivers* will be loaded exactly as before.
- Modify the periodic tasks collection code to also collect periodic tasks for enabled interfaces of every enabled *hardware type*.
- Conductor will fail to start if there is a name clash between a *classic driver* and a *hardware type*.

Database and Rest API

- Allow the node `driver` field to accept the *hardware types* as well. This will work in all API versions.

  Note

  There are two reasons for that:

  – Consistency: we never prevented new drivers to be used with old API versions, and *dynamic drivers* will look mostly like new drivers to users.

  – Usability: we plan on eventually deprecating the classic drivers. When we remove them, all clients will need to specify the *hardware types* when enrolling nodes. To allow older clients to continue interacting with the API service, even as they use new driver names (hardware types), we must continue to use the same field name and API semantics.

- For each interface create a new field on the `node` table named `<interface_name>_interface`. A migration will be needed each time we add a new interface (which hopefully wont happen too often).

  For *hardware types* setting `<interface_name>_interface` field to `None` means using the calculated default value as described in *Configuration*.

  Trying to set any of these fields to a value other than `None` will result in an error if the `driver` field is set to a *classic driver*. Similarly, all these fields are reset to `None` if the `driver` field is set to a *classic driver*.

- Every time `driver` and/or any of the interface fields is updated, the conductor checks that the *hardware type* supports all the resulting interfaces (except when `driver` is set to a *classic driver*).

  To change between two incompatible sets of interfaces, all changes should come in one API call. E.g. for a node with the `ilo-gen8` *hardware type* and `vmedia_ilo` boot interface the following JSON patch will be allowed:

  ```
  [
      {"op": "replace", "path": "/boot_interface", "value": "ipxe"},
      {"op": "replace", "path": "/driver", "value": "generic-ipmi"}
  ]
  ```

  but the following patch will fail because of incompatible boot interface:

  ```
  [
      {"op": "replace", "path": "/driver", "value": "generic-ipmi"},
  ]
  ```

  Note

  RFC 6902 requires a JSON patch to be atomic, because an HTTP PATCH operation must be atomic. Meaning, its possible for some operations to end up with an inconsistent object as long as the end result is consistent.

  The validation will be conducted on the API service side by checking the new

conductor_hardware_interfaces database table.

- If for some reason the existing *interface* becomes invalid for a node (e.g. it was disabled after the node was enrolled), it will be signalized via the usual node validation API. The validation for this interface wont pass with an appropriate error message. On the programming level, the driver attribute for this interface (e.g. `task.driver.deploy`) will be set to `None`.
- Update `GET /v1/drivers` to also list enabled *hardware types*. This change is **not** affected by API versioning, because we allow old API versions to use *hardware types* with the `driver` field.
- Allow `GET /v1/drivers` to filter only *hardware types* or only *classic drivers*.

  Update `GET /v1/drivers/<HW TYPE>` to report the *hardware type* information, including the list of enabled *hardware interfaces*.

  This feature is guarded by an API version bump (as usual).
- Allow filtering nodes by `<interface_name>_interface` fields in the node list API.

  This feature is guarded by an API version bump (as usual).
- Create a new table `conductor_hardware_interfaces` to hold the relationship between conductors, hardware types and available interfaces. A warning will be issued on conductor start up, if it detects that other conductors have a different set of interfaces for the same enabled *hardware type*. This will also track the default interface for each hardware type and interface type combination.

  This situation is inevitable during live upgrades, so it must not result in an error. However, we will document that all conductors should have the same set of interfaces for the same enabled *hardware types*.

  This table will not be exposed in the HTTP API for now.

Deprecations

We are **not** planning to deprecate and remove the support for *classic drivers* in the V1 API.

We are planning to deprecate and remove the *classic drivers* which exist in-tree. The deprecation procedure may be tricky and will be covered by a follow-up spec.

Alternatives

- We could put interfaces under a new JSON key on a node. However, were trying to move away from informally defined JSON keys. It would also prevent us from being able to implement the filtering of nodes based on a particular interface.
- We could create a new API endpoint for updating the interfaces. This will be inconsistent with how we update the `driver` field though.

  We could then create a new API version, preventing updating `driver` via the regular node update API, but that would be a breaking change.
- We could create a new field `hardware_type` instead of having the existing `driver` field accept a *hardware type*. This was a part of the proposal previously, but we found that it complicates things substantially without clear benefits.
- We could create a whole new family of API endpoints instead of reusing `/v1/drivers`, e.g. `/v1/hardware-types`. However, it would require us to replicate all driver-related functionality nearly intact, for example driver vendor passthru. So users would have to somehow figure out which vendor passthru endpoint to use based on what kind of a driver is in the `driver` field.

Data model impact

- For each interface, create a new node field `<interface_name>_interface` initially set to `NULL`.
- Create a new internal table `conductor_hardware_interfaces`:

  `conductor_id` - conductor ID (foreign key to conductors table),

  `hardware_type VARCHAR(255)` - *hardware type* entrypoint name,

  `interface_type VARCHAR(16)` - interface type name (e.g. `deploy`),

  `interface_name VARCHAR(255)` - interface implementation entry point name.

  `default TINYINT(1)` - boolean which denotes if this `interface_name` is
  the default for a given `hardware_type` and `interface_type` combination.

  This table will get populated on conductor start up and purged on deleting the conductor record. On conductor startup, during init_host(), the conductor will fetch the list of hardware interfaces supported by all registered conductors and compare to its own configuration. If the same *hardware type* is enabled on two conductors with a different set of enabled_interfaces, this will result in a WARNING log message. The enabled *hardware types* themselves do not have to match (just like today, different conductors can have different set of drivers).

State Machine Impact

None

REST API impact

- Update `GET /v1/drivers`:

  Return both *classic drivers* and *hardware types* no matter which API version is used.

  New URL parameters:

  - `type` (string, one of `classic`, `dynamic`, optional) - if provided, limit the resulting driver list to only *classic drivers* or *hardware types* accordingly.

  New response field:

  `type` whether the driver is *dynamic* or *classic*.

  This change is guarded by a new API version.

- Update `GET /v1/drivers/<NAME>`:

  New response field:

  `type` whether the driver is *dynamic* or *classic*.

  New response fields that are not `None` only for *hardware types*:

  `default_<interface_name>_interface`
  the entrypoint name of the calculated default implementation for a given interface.

  `enabled_<interface_name>_interfaces`
  the list of entrypoint names of enabled implementations for a given interface.

- Update `GET /v1/drivers/<NAME>/properties` and `GET /v1/drivers/<NAME>/vendor_passthru/methods` and the actual driver vendor passthru call implementation:

When requested for a *dynamic driver*, assume the calculated defaults for the `vendor` interface implementation as described in *Configuration*. We will need to support non-default implementations as well, but it goes somewhat beyond the scope of this already big spec.

Client (CLI) impact

ironic CLI

- Update the node creation command to accept one argument per interface. Example:

```
ironic node-create --driver=ilo-gen9 --power-interface=redfish
```

  The same change is applied to the OSC plugin.

- Extend the output of the `driver-list` command with the `Type` column.
- Extend the `driver-list` command with `--type` argument, which, if supplied, limits the driver list to only *classic drivers* (`classic` value) or *hardware types* (`dynamic` value).
- Extend the output of the `driver-show` command with the newly introduced fields.

openstack baremetal CLI

Similar changes to whats in *ironic CLI* are applied here.

RPC API impact

- No impact on the hash ring, as both *hardware types* and *classic drivers* are used in the same field.

Driver API impact

Hardware Types

- Create a new `AbstractHardwareType` class as an abstract base class for all hardware types. Here is a simplified example implementation, using only power, deploy and inspect interfaces:

```
import abc, six

@six.add_metaclass(abc.ABCMeta)
class AbstractHardwareType(object):
    @abc.abstractproperty
    def supported_power_interfaces(self):
        pass

    @abc.abstractproperty
    def supported_deploy_interfaces(self):
        pass

    @property
    def supported_inspect_interfaces(self):
        return [NoopInspect]
```

  Note that some interfaces (power, deploy) are mandatory, while the other (inspect) are not. A dummy implementation will be provided for all optional interfaces. Depending on the specific call

it will either do nothing or raise an error. For user-initiated calls (e.g. start inspection), an error will be returned. For internal calls (e.g. attach cleaning ports), no action will be taked.

- Create a new `GenericHardwareType` class which most of the actual hardware type classes will want to subclass. This class will insert generic implementations for some interfaces:

```
class GenericHardwareType(AbstractHardwareType):
    supported_deploy_interfaces = [AgentDeploy]
    supported_inspect_interfaces = [NoopInspect, InspectorInspect]
```

Note that all properties contain classes, not instances. Also note that order matters: in this example `NoopInspect` will be the default, if both implementations are enabled in the configuration.

- Here is an example of how hardware types could be created:

```
class GenericIpmiHardware(GenericHardwareType):
    supported_power_interfaces = [IpmitoolPower, IpminativePower]

class iLOGen8Hardware(GenericHardwareType):
    supported_power_interfaces = (
        GenericIpmiHardware.supported_power_interfaces
        + [IloPower]
    )
    supported_inspect_interfaces = (
        GenericHardwareType.supported_inspect_interfaces
        + [IloInspect]
    )

class iLOGen9Hardware(iLOGen8Hardware):
    supported_power_interfaces = (
        iLOGen8Hardware.supported_power_interfaces
        + [RedfishPower]
    )
```

Note

These definitions use classes, not entrypoints names. These examples assume the required classes are imported.

Note

The following entrypoints will have to be defined for these examples to work:

```
ironic.hardware.types =
    generic-ipmi = ironic.hardware.ipmi:GenericIpmiHardware
    ilo-gen8 = ironic.hardware.ilo:iLOGen8Hardware
    ilo-gen9 = ironic.hardware.ilo:iLOGen9Hardware

ironic.hardware.interfaces.power =
    ipmitool = ironic.drivers.modules.ipmitool:IpmitoolPower
    ipminative = ironic.drivers.modules.ipmitool:IpminativePower
```

```
    ilo = ironic.drivers.modules.ilo:IloPower
    redfish = ironic.drivers.modules.redfish:RedfishPower

ironic.hardware.interfaces.inspect =
    inspector = ironic.drivers.modules.inspector:InspectorInspect
    ilo = ironic.drivers.modules.ilo:IloInspect
```

The following configuration will be required to enable everything in these examples:

```
[DEFAULT]
enabled_hardware_types = generic-ipmi,ilo-gen8,ilo-gen9
enabled_power_interfaces = ipmitool,ipminative,ilo,redfish
enabled_inspect_interfaces = inspector,ilo
```

Driver Creation

- At start up time the conductor instantiates all enabled hardware types, as well as all enabled interface implementations for enabled hardware types.
- Each time the node is created or loaded from the database, a thin BareDriver object is created with all interfaces set on it. This is similar to how network drivers already work. It gets assigned to `task.driver`, and after that everything works as before this spec.

Nova driver impact

None

Ramdisk impact

None

Security impact

None

Other end user impact

- End users should switch to *hardware types* over time.

Scalability impact

None

Performance Impact

- A driver instance will be now created per node as opposed to creating one per conductor right now. This will somewhat increase the memory usage per node. We can probably define __slots__ on the driver class to reduce this effect.

Other deployer impact

- A deployer can set the new `enabled_hardware_types` option to enable more *hardware types*. Otherwise only the default *hardware types* and already enabled classic drivers will be available.
- A deployer can also set any of new `enabled_<INTERFACE>_interfaces` options to enable more *interfaces* for the enabled *hardware types*.

Developer impact

This spec changes the way we expect the developers to write their drivers.

- No more new *classic drivers* will be accepted in-tree as soon as this change lands.
- Developers should implement *hardware types* and *interfaces* to provide new hardware support for Ironic. Built-in *interfaces* implementations will be available for reuse both in-tree and out-of-tree.

Implementation

Assignee(s)

- Dmitry Tantsur (lp: divius, irc: dtantsur)
- Jim Rollenhagen (irc: jroll)

Work Items

- Create base classes supporting *hardware types*.
- Create tables for tracking enabled *hardware interfaces*.
- Load *hardware types* on conductor start up and record them in the internal table.
- Create node fields for *interfaces* and expose them in the API.
- Update the drivers API to support *hardware types*.
- Create the *hardware types* for hardware supported directly by the team, i.e. the generic IPMI-compatible hardware. The SSH driver might be removed soon; it wont get updated in this case.

Dependencies

- For the vendor interface to be really pluggable, we need to promote agent passthru to the core API.

Testing

- Unit test coverage will obviously be provided.
- A new gate job will be created, using a dynamic version of the IPMI driver. We will aim to make it the primary approach in the gate over time.
- Grenade testing for upgrades / migration of existing workloads to new drivers.

Upgrades and Backwards Compatibility

This reform is designed to be backward compatible. The *classic drivers* will be supported for at least some time. A separate spec will cover the deprecation of the *classic drivers*.

We will recommend switching to using appropriate *dynamic drivers* as soon as its possible.

Upgrade flow

1. Ironic is updated to a version supporting *dynamic drivers*. The API version used by clients is not updated yet.
2. All nodes are still using *classic drivers*. On a node `driver=x_y`.
3. Users with an old API version:
   - can set `driver` to a *classic driver*.
   - can set `driver` to a *hardware type*, which will result in using a *dynamic driver* with the default set of interfaces.
4. Users with a new API version:
   - can set `driver` to a *hardware type* or a *classic driver*
   - can set non-default interface implementations when `driver` is set to a real *hardware type*

Documentation Impact

- Document switching to *dynamic drivers*
- Document creating new *hardware types*

References

Initial etherpad: https://etherpad.openstack.org/p/liberty-ironic-driver-composition

Newton etherpad: https://etherpad.openstack.org/p/ironic-newton-summit-driver-composition

Enhance Power Interface for Soft Power Off and NMI

https://bugs.launchpad.net/ironic/+bug/1526226

The proposal presents the work required to enhance the power interface to support soft reboot and soft power off, and the management interface to support diagnostic interrupt (NMI [1]).

Problem description

There exists a problem in the current driver interface which doesnt provide with soft power off and diagnostic interrupt (NMI [1]) capabilities even though ipmitool [2] and most of BMCs support these capabilities.

Here is a part of ipmitool man page in which describes soft power off and diagnostic interrupt (NMI [1]).

$ man ipmitool:

```
...
power

      Performs a chassis control command to view and change the
      power state.

      ...

      diag

            Pulse a diagnostic interrupt (NMI) directly to the
            processor(s).

      soft

            Initiate a soft-shutdown of OS via ACPI. This can be
            done in a number of ways, commonly by simulating an
            overtemperature or by simulating a power button press.
            It is necessary for there to be Operating System
            support for ACPI and some sort of daemon watching for
            events for this soft power to work.
```

From customers point of view, both tenant admin and tenant user, the lack of the soft power off and diagnostic interrupt (NMI [1]) lead the following inconveniences.

1. Customer cannot safely shutdown or soft power off their instance without logging on.
2. Customer cannot take NMI dump to investigate OS related problem by themselves.

From deployers point of view, that is cloud provider, the lack of the two capabilities leads the following inconveniences.

1. Cloud provider support staff cannot shutdown customers instance safely without logging on for hardware maintenance reason or etc.
2. Cloud provider support staff cannot ask customer to take NMI dump as one of investigation materials.

Proposed change

In order to solve the problems described in the previous section, this spec proposes to enhance the power states, the PowerInterface base class and the ManagementInterface base class so that each driver can implement to initiate soft reboot, soft power off and inject NMI.

And this enhancement enables the soft reboot, soft power off and inject NMI through Ironic CLI and REST API for tenant admin and cloud provider. Also this enhancement enables them through Nova CLI and REST API for tenant user when Novas blueprint [3] is implemented.

As a reference implementation, this spec also proposes to implement the enhanced PowerInterface base class into the IPMIPower concrete class and the enhanced ManagementInterface base class into the IPMIManagement concrete class.

1. add the following new power states to ironic.common.states:

```
SOFT_REBOOT = 'soft rebooting'
SOFT_POWER_OFF = 'soft power off'
```

2. add get_supported_power_states method and its default implementation to the base PowerInterface class in ironic/drivers/base.py:

```
def get_supported_power_states(self, task):
    """Get a list of the supported power states.

    :param task: A TaskManager instance containing the node to act on.
    :returns: A list of the supported power states defined
              in :mod:`ironic.common.states`.
    """
    return [states.POWER_ON, states.POWER_OFF, states.REBOOT]
```

   - Note: WakeOnLanPower driver supports only states.POWER_ON.

3. add a default parameter `timeout` into the set_power_state method in to the base PowerInterface class in ironic/drivers/base.py:

```
@abc.abstractmethod
def set_power_state(self, task, power_state, timeout=None):
    """Set the power state of the task's node.

    :param task: a TaskManager instance containing the node to act on.
    :param power_state: Any power state from :mod:`ironic.common.states`.
    :param timeout: timeout positive integer (> 0) for any power state.
       ``None`` indicates to use default timeout which depends on
       ``power_state``[*]_ and driver.
    :raises: MissingParameterValue if a required parameter is missing.
    """
```

4. enhance set_power_state method in IPMIPower class so that the new states can be accepted as power_state parameter.

   IPMIPower reference implementation supports SOFT_REBOOT and SOFT_POWER_OFF.

   SOFT_REBOOT is implemented by first SOFT_POWER_OFF and then a plain POWER_ON such that Ironic implemented REBOOT. This implementation enables generic BMC detect the reboot completion as the power state change from ON -> OFF -> ON which power transition is called `power cycle`.

   The following table shows power state value of each state variables. `new_state` is a value of the second parameter of set_power_state() function. `power_state` is a value of node property. `target_power_state` is a value of node property.

new_state	power_state (start state)	target_power_state (assigned value)	power_state (end state)
SOFT_REBOOT	POWER_ON	SOFT_POWER_OFF	POWER_OFF[*]_
SOFT_REBOOT	POWER_OFF[*]_	POWER_ON	POWER_ON
SOFT_POWER_OFF	POWER_OFF	POWER_ON	POWER_ON
SOFT_POWER_OFF	POWER_ON	SOFT_POWER_OFF	POWER_OFF
	POWER_OFF	NONE	POWER_OFF

new_state	power_state (start state)	target_power_state (assigned value)	power_state (end state)
SOFT_REBOOT	POWER_ON	SOFT_POWER_OFF	ERROR[*]_
SOFT_POWER_OFF	POWER_ON	SOFT_POWER_OFF	ERROR[*]_

5. add get_supported_power_states method and implementation in IPMIPower:

```
def get_supported_power_states(self, task):
    """Get a list of the supported power states.

    :param task: A TaskManager instance containing the node to act on.
       currently not used.
    :returns: A list of the supported power states defined
              in :mod:`ironic.common.states`.
    """

    return [states.POWER_ON, states.POWER_OFF, states.REBOOT,
            states.SOFT_REBOOT, states.SOFT_POWER_OFF]
```

6. add inject_nmi abstract method to the base ManagementInterface class in ironic/drivers/base.py:

```
@abc.abstractmethod
def inject_nmi(self, task):
    """Inject NMI, Non Maskable Interrupt.

    :param task: A TaskManager instance containing the node to act on.
    :returns: None
    """
```

7. add inject_nmi concrete method implementation in IPMIManagement class.

Alternatives

- Both the soft power off and diagnostic interrupt (NMI [1]) could be implemented by vendor passthru. However the proposed change is better than the vendor passthru, because users of Ironic API or Ironic CLI can write script or program uniformly.

Data model impact

None

State Machine Impact

None

REST API impact

- Add support of SOFT_REBOOT and SOFT_POWER_OFF to the target parameter of following API:

```
PUT /v1/nodes/(node_ident)/states/power

The target parameter supports the following JSON data respectively.
``timeout`` is an optional parameter for any ``target`` parameter.
In case of "soft reboot" and "soft power off", ``timeout`` overrides
``soft_power_off_timeout`` in the in the Ironic configuration file,
typically /etc/ironic/ironic.conf.

Examples

  {"target": "soft reboot",
   "timeout": 900}

  {"target": "soft power off",
   "timeout": 600}
```

- Add a new management API to support inject NMI:

```
PUT /v1/nodes/(node_ident)/management/inject_nmi

Request doesn't take any parameter.
```

Client (CLI) impact

- Enhance Ironic CLI ironic node-set-power-state to support power graceful off/reboot by adding optional arguments. This CLI is async. In order to get the latest status, call ironic node-show-states and check the returned value.:

```
usage: ironic node-set-power-state <node> <power-state>
       [--soft] [--timeout <timeout>]

Power a node on/off/reboot, power graceful off/reboot to a node.

Positional arguments

<node>

    Name or UUID of the node.

<power-state>

    'on', 'off', 'reboot'

Optional arguments:
   --soft
     power graceful off/reboot.

   --timeout <timeout>
     timeout positive integer value(> 0) for any ``power-state``.
     If ``--soft`` option is also specified, it overrides
```

(continues on next page)

(continued from previous page)

```
    ``soft_power_off_timeout`` in the in the Ironic configuration
    file, typically /etc/ironic/ironic.conf.
```

- Add a new Ironic CLI ironic node-inject-nmi to support inject nmi. This CLI is async. In order to get the latest status, serial console access is required.:

```
usage: ironic node-inject-nmi <node>

Inject NMI, Non Maskable Interrupt.

Positional arguments

<node>

    Name or UUID of the node.
```

- Enhance OSC plugin openstack baremetal node so that the parameter can accept reboot [soft] [timeout <timeout>], power [on|off [soft] [timeout <timeout>] and inject nmi. This CLI is async. In order to get the latest status, call openstack baremetal node show and check the returned value.:

```
usage: openstack baremetal node reboot [--soft] [--timeout <timeout>]
↪<uuid>

usage: openstack baremetal node power off [--soft] [--timeout <timeout>]
↪<uuid>

usage: openstack baremetal node inject nmi <uuid>
```

RPC API impact

None

Driver API impact

PowerInterface base and ManagementInterface base are enhanced by adding a new method respectively as described in the section Proposed change. And these enhancements keep API backward compatible. Therefore it doesnt have any risk to break out of tree drivers.

Nova driver impact

The default behavior of nova reboot command to a virtual machine instance such as KVM is soft reboot. And nova reboot command has a option hard to indicate hard reboot.

However the default behavior of nova reboot to an Ironic instance is hard reboot, and hard option is meaningless to the Ironic instance.

Therefore Ironic Nova driver needs to be update to unify the behavior between virtual machine instance and bare-metal instance.

This problem is reported as a bug [6]. How to fix this problem is specified in nova blueprint [10] and spec [11].

The default behavior change of nova reboot command is made by following the standard deprecation policy [12]. How to deprecate nova command is also specified in nova blueprint [10] and spec [11].

Ramdisk impact

None

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

- Deployer, cloud provider, needs to set up ACPI [7] and NMI [1] capable bare metal servers in cloud environment.
- change the default timeout value (sec) in the Ironic configuration file, typically /etc/ironic/ironic.conf if necessary.

Developer impact

- Each driver developer needs to follow this interface to implement this proposed feature.

Implementation

Assignee(s)

Primary assignee:
Naohiro Tamura (naohirot)

Other contributors:
None

Work Items

- Enhance PowerInterface class and ManagementInterface class to support soft power off and inject nmi [1] as described Proposed change.
- Enhance Ironic API as described in REST API impact.
- Enhance Ironic CLI as described in Client (CLI) impact.

- Implement the enhanced PowerInterface class into the concrete class IPMIPower, and the enhanced ManagementInterface class into the concrete class IPMIManagement. Implementing vendors concrete class is up to each vendor.
- Coordinate the work with Nova NMI support Inject NMI to an instance [3] if necessary.
- Update the deployer documentation from the ironic perspective.

Dependencies

- Soft power off control depends on ACPI [7]. In case of Linux system, acpid [8] has to be installed. In case of Windows system, local security policy has to be set as described in Shutdown: Allow system to be shut down without having to log on [9].
- NMI [1] reaction depends on Kernel Crash Dump Configuration. How to set up the kernel dump can be found for Linux system in [13], [14], and for Windows in [15].

Testing

- Unit Tests.
- Tempest Tests, at least soft reboot/soft power off.
- Each vendor plans Third Party CI Tests if implemented.

Upgrades and Backwards Compatibility

None (Forwards Compatibility is out of scope)

- Note The backwards compatibility issue of the default behavior change of nova reboot command is solved by following the standard deprecation policy [12].

Documentation Impact

- The deployer doc and REST API reference manual need to be updated. (CLI manual is generated automatically from source code)

References

[1] http://en.wikipedia.org/wiki/Non-maskable_interrupt

[2] http://linux.die.net/man/1/ipmitool

[3] https://review.opendev.org/#/c/187176/

[4] https://en.wikipedia.org/wiki/Communicating_sequential_processes

[5] http://linux.die.net/man/1/virsh

[6] https://bugs.launchpad.net/nova/+bug/1485416

[7] http://en.wikipedia.org/wiki/Advanced_Configuration_and_Power_Interface

[8] http://linux.die.net/man/8/acpid

[9] https://technet.microsoft.com/en-us/library/jj852274%28v=ws.10%29.aspx

[10] https://blueprints.launchpad.net/nova/+spec/soft-reboot-poweroff

[11] https://review.opendev.org/#/c/229282/

[12] http://governance.openstack.org/reference/tags/assert_follows-standard-deprecation.html

[13] https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/html/Kernel_Crash_Dump_Guide/

[14] https://help.ubuntu.com/lts/serverguide/kernel-crash-dump.html

[15] https://support.microsoft.com/en-us/kb/927069

Interface Attach and Detach API

https://bugs.launchpad.net/ironic/+bug/1582188

We are adding pluggable network interfaces to Ironic. These interfaces will be configurable per Ironic node, this means that different nodes might have different ways of handling their network interfaces. We need a way that different network interfaces can override the virtual network interface (VIF) to physical network interface (PIF) mapping logic currently contained in nova.

Problem description

Currently we use an Ironic port update to assign tenant neutron ports to Ironic ports using their extra field. Doing the mapping this way may not work for a third party network interface implementation. This also ties the ironic nova virt driver to a particular network VIF to PIF mapping implementation. If a third party network interface wants to do something different with network VIF to PIF mapping such as storing the port IDs for dynamic vNIC creation later in the provisioning process or we as the Ironic team want to change an implementation detail we have to submit changes into nova. Additionally if we want to support post-deployment attach and detach of a network VIF then we have to watch for any updates made to the port object and interpret certain changes as certain actions.

Proposed change

To solve this problem I propose to add a new API endpoint,

- POST v1/nodes/<node_id>/vifs
- GET v1/nodes/<node_id>/vifs
- DELETE v1/nodes/<node_id>/vifs/<vif_id>

These API endpoints will take via a POST body, a JSON representation of a generic VIF object. Making it generic allows for non-neutron based implementations to use this API. This VIF object will be passed to new functions in the pluggable network interfaces:

def vif_attach(self, vif):

def vif_detach(self, vif_id):

def vif_list(self):

The network interface can use these functions to handle attaching the VIF to the Ironic node in whichever way it needs to for its implementation. This could be by adding a field to the Ironic port as with the existing implementation, or it might be different for example storing it in a list in the nodes driver_internal_info.

The ironic nova virt driver will be updated to use this new API in the plug_vifs and unplug_vifs functions, unbinding it from the underlying implementation details.

Alternatives

- Continue to use a port update to allow nova to interact with ironic ports and soon portgroups, documenting the vif_port_id etc logic as a defined API for the network interfaces to use.

Data model impact

None

State Machine Impact

None

REST API impact

- GET v1/nodes/<node_id>/vifs
  - Calls to vif_list on the nodes network interface
  - This is a synchronous API endpoint, and the normal ironic rules apply for avoiding implementing tasks that take a long time or are unstable under synchronous endpoints.
  - Method: GET
  - Successful http response: 200
  - Expected response body is a JSON
    - JSON Schema:

```
{
    "title": "VIFS",
    "type": "object",
    "properties": {
        "vifs": {
            "description": "List of VIFs currently attached"
            "type": "array",
            "items": {
                "type": "object",
                "properties": {
                    "id": {
                        "type": "string"
                    }
                },
                "required": ["id"]
            }
        }
    },
    "required": ["vifs"]
}
```

    - Example:

```
{
    "vifs": [
                {
                    "id": "8e6ba175-1c16-4dfa-82b9-dfc12f129170",
                }
    ]
}
```

- POST v1/nodes/<node_id>/vifs
  - Calls vif_attach on the network interface for the node, passing in the json provided
  - This is a synchronous API endpoint, and the normal ironic rules apply for avoiding implementing tasks that take a long time or are unstable under synchronous endpoints.
  - Method: POST
  - Successful http response: 204
  - Expected error response codes:
    - 404, Node not found
    - 400, Request was malformed
    - 409, Conflict between the requested VIF to attach and other VIFs already attached
    - 422, The request was good but unable to attach the VIF for a defined reason, for example: No physical interfaces available to attach too
  - Expected data is a JSON
    - JSON Schema:

```
{
    "title": "Attachment",
    "type": "object",
    "properties": {
        "id": {
            "description": "ID of VIF to attach"
            "type": "string"
        },
    },
    "required": ["id"]
}
```

    - Example:

```
{
  "id": "8e6ba175-1c16-4dfa-82b9-dfc12f129170"
}
```

  - Expected response body is empty
- DELETE v1/nodes/<node_id>/vifs/<vif_id>
  - Calls vif_detach on the network interface for the node, passing in the json provided

  - This is a synchronous API endpoint, and the normal ironic rules apply for avoiding implementing tasks that take a long time or are unstable under synchronous endpoints.
  - Method: DELETE
  - Successful http response: 204
  - Expected error response codes:
    * 404, Node not found
    * 400, Request was malformed
    * 422, The request was good but unable to detach the VIF for a defined reason
  - Expected response body is empty.
- Does the API microversion need to increment? Yes
- Is a corresponding change in the client library and CLI necessary? Yes
- As these are new API entry points they will not affect older clients.

Client (CLI) impact

ironic CLI

- ironic node-vif-list <node_id>
- ironic node-vif-attach <node_id> <vif_id>
- ironic node-vif-detach <node_id> <vif_id>

openstack baremetal CLI

- openstack baremetal node vif list <node_id>
- openstack baremetal node vif attach <node_id> <vif_id>
- openstack baremetal node vif detach <node_id> <vif_id>

RPC API impact

The RPC API will implement:

- vif_attach(self, context, node_id, vif)
- vif_detach(self, context, node_id, vif_id)
- vif_list(self, context, node_id)

Driver API impact

Base network interface will need to be extended with:

```
def vif_list(self, task):
    # TODO(sambetts): Uncomment when vif_port_id in port.extra is removed.
    # raise NotImplemented
    default_vif_list()
```

(continues on next page)

(continued from previous page)

```
def vif_attach(self, task, vif):
    # TODO(sambetts): Uncomment when vif_port_id in port.extra is removed.
    # raise NotImplemented
    default_vif_attach(vif)

def vif_detach(self, task, vif_id):
    # TODO(sambetts): Uncomment when vif_port_id in port.extra is removed.
    # raise NotImplemented
    default_vif_detach(vif_id)
```

Existing flat, neutron and noop network interfaces will need extending to include implementations for these functions.

Flat network driver will need to implement add_provisioning_network to bind the ports that used to be bound by nova.

Nova driver impact

plug/unplug_vifs logic will be replaced by calling attach/detach for every VIF passed into those functions.

nova.virt.driver.IronicDriver.macs_for_instance will be removed because mapping is handled inside Ironic so mac_address assignment must happen at binding later in the process.

nova.virt.driver.IronicDriver.network_binding_host_id will be changed to return None in all cases, so that neutron ports remain unbound until Ironic binds them during deployment.

nova driver will need to include the nova compute host ID in the instance_info so that the ironic flat network interface can use it to update the neutron ports mimicking the existing nova behavior.

The nova driver ironic API version requirement will need to be increased to the version that implements the attach and detach APIs. Operators will need to ensure the version of python-ironicclient they have installed on the nova compute service supports the new APIs.

Ramdisk impact

None

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

None

Developer impact

Developers of network interfaces will need to consider how their network interface wants to handle ports.

Implementation

Assignee(s)

Primary assignee:
sambetts

Work Items

- Add new APIs to Ironic
- Update existing ironic network interfaces to support the new APIs
- Add new APIs to ironic client
- Update nova virt driver to use new APIs via client

Dependencies

- Ironic neutron integration
  - https://specs.openstack.org/openstack/ironic-specs/specs/not-implemented/ironic-ml2-integration.html

Testing

These changes will be tested as part of the normal gate process, as they will be part of the normal ironic deployment workflow.

Upgrades and Backwards Compatibility

Setting vif_port_id via a port/portgroup update will be deprecated in favor of the new APIs. A deprecation message should be issued on a port update if a user is directly setting vif_port_id on a port or portgroup. Code will need to be added to ensure that the network interfaces will still process a vif_port_id set via that method.

As the old vif_port_id field is still supported although deprecated, old nova virt drivers will continue to work using that method.

Ironic must be upgraded before Nova is as newer nova virt drivers wont work with older versions of Ironic which are missing the new APIs.

Out-of-tree network interfaces may not immediately implement the interface_attach and interface_detach methods, so during the period that vif_port_id is a deprecated method, we should also ensure we provide

a default implementation of attach and detach, this default implementation should match the behaviour of ironic nova virt driver implementation setting the vif_port_id in port.extra and will be removed when support for vif_port_id in port.extra is removed.

Documentation Impact

New API and its usage needs to be documented.

References

None

Ironic Neutron Integration

https://bugs.launchpad.net/ironic/+bug/1526403 https://bugs.launchpad.net/ironic/+bug/1618754

The current Ironic implementation only supports flat networks. For isolation of tenant networks Ironic should be able to pass information to Neutron to allow for provisioning of the baremetal server onto the tenant network.

Problem description

Ironic currently provisions servers only on flat networks hence providing no network isolation between tenants. Ironic should allow end users to utilize a baremetal instance in the same isolated (e.g. VLAN, VXLAN) networks as their virtual machines are.

In order to provide the required network isolation, Ironic should be able to provide the requisite connectivity information (using LLDP) to the Neutron ML2 plugin to allow drivers to provision the top-of-rack (ToR) switch for the baremetal server.

Ironic also poses new challenges for Neutron, for example, the concepts of link aggregation and multiple networks per node. This spec covers the concept of link aggregation where multiple interfaces on the bare metal server connect to switch ports of a single LAG and belong to the same network. However,this spec does not cover:

- the case of trunked ports belonging to multiple networks - this will need a separate spec and implementation in Neutron, see vlan-aware-vms.
- the case where the bare metal server has multiple interfaces that belong to different networks. This may require the capability to specify which NICs belong to which network. It is in scope but the method of deciding which interface gets which network will be decided elsewhere.

Proposed change

When nodes are enrolled/introspected in Ironic, local link (e.g. LLDP) information should be recorded for each port. The option of creating a portgroup to specify an aggregation of said ports should be made available.

Ironic should then send the local link information to Neutron in the form of a port binding profile. For each portgroup, the port binding profile will contain a list of the local link information for each port in that portgroup. For each port not belonging to a portgroup, the port binding profile will contain only the local link information for that particular port. Therefore the length of the list of local link information can be used by Neutron to determine whether to plumb the network for a single port or for a portgroup (in the case of a portgroup there may be additional switch configurations necessary to manage the associated LAG). Each port binding profile sent to the Neutron port create API will result in the creation of a single

Neutron port. In this way, each portgroup can represent a LAG for which all member switch ports will belong to the same network and to the same network segment.

The node should first be placed on a provisioning network and then onto the tenant network as specified in the network-provider spec[1]. To help facilitate this there will be a variable stored in the binding profile to record whether a port binding has been requested. This will allow Ironic to defer binding until Ironic is able to populate the extra information that is required by Neutron.

Note also, there will be no support for PXE booting after deploy (i.e. local disk installation only). The reason for this is because the nodes will not be able to PXE boot if they cannot reach the TFTP server. Local boot will need to be used for any node deployed outside of the provisioning network. Instance netboot should not be permitted when this feature is used. This limitation could be addressed if routing was set up from every tenant network to the TFTP server or if Ironic would work with a TFTP server per tenant network, but these options are out of scope in this spec.

Note

It may be possible for virtual media drivers to support both netboot and localboot when provisioning/tenant networks are isolated. This is because virtual media drivers boot the bare metal using out of band means using BMC and hence bare metal servers NICs dont require to access Ironic conductor when booting the instance.

A supportive ML2 driver can then use the information supplied to provision the port.

The Ironic port object will be updated with a new local_link_connection field dict supplying the following values:

- switch_id
- port_id
- switch_info

These fields are in string format. The information could be gathered by using LLDP based on the LLDP TLV counterparts of chassis_id, port_id and system_name, although it is not strictly required to populate these fields with LLDP values. For example, the switch_id field is used to identify a switch and could be an LLDP based MAC address or an OpenFlow based datapath_id. The switch_info field could be used to distinguish different switch models or some other vendor specific identifier. The switch_id and port_id fields are required and the switch_info field can be optionally populated. The key point is that Ironic and Neutron should share the same understanding of this information in order to connect the Ironic instance to the appropriate Neutron network.

To facilitate link aggregation a new portgroup object will be created. In addition to the base object it will have the following fields:

- id
- uuid
- name
- node_id
- address
- mode

[1] http://specs.openstack.org/openstack/ironic-specs/specs/6.1/network-provider.html

- properties
- extra
- internal_info
- standalone_ports_supported

The `address` field represents the MAC address for bonded NICs of the bare metal server. It is optional, as its value is highly dependent on the instance OS. In case of using ironic through nova, the `address` gets populated by whatever value is generated by neutron for the VIFs address representing this portgroup, if there is one. This happens during every VIF attach call. In case of standalone mode, it can be always `None`, as it should be configured on the instance through the configdrive.

The `mode` field is used to specify the bond mode. If not provided in the portgroup creation request, its default value is determined by the `[DEFAULT]default_portgroup_mode` configuration option, which in turn has a default value of `active-backup`. For a portgroup that was created in an older API version, this configuration value will be used as the value for that portgroups mode. `active-backup` is chosen as the default because this mode does not require any additional configuration on the switch side.

The `properties` field is a dictionary that can contain any parameters needed to configure the portgroup.

For additional information about `mode` and `properties` fields contents, see linux kernel bonding documentation.

The `extra` field can be used to hold any additional information that operators or developers want to store in the portgroup.

The `internal_info` field is used to store internal metadata. This field is read-only.

The `standalone_ports_supported` indicates whether ports that are members of this portgroup can be used as stand-alone ports.

The Ironic port object will then have the following fields added to support new functionality:

- local_link_connection
- portgroup_id
- pxe_enabled

If there are multiple pxe_enabled ports or portgroups, dhcpboot options will be set for all portgroups and all pxe_enabled ports not belonging to any portgroup.

The following port binding related information needs to be passed to Neutron:

Field Name	Description
vnic_type	Type of the profile (baremetal in this case). This would allow at least basic filtering for Ironic ports by ML2 drivers.
local_link_infor	A list of local link connection information either from all Ironic ports in a particular portgroup or from a single Ironic port not belonging to any portgroup.
host_id	This should be set to the Ironic node uuid.

A JSON example to describe the structure is:

```
{
  "port": {
```

(continues on next page)

(continued from previous page)

```
    <all other fields>,
    "vnic_type": "baremetal",
    "host_id": <Ironic node UUID>,
    "binding:profile": {
      "local_link_information": [
        {
          "switch_id": xxx,
          "port_id": xxx,
          "switch_info": zzz,
          <optional more information>
        },
        {
          "switch_id": xxx,
          "port_id": yyy,
          "switch_info": zzz,
          <optional more information>
        }
      ]
      <some more profile fields>
    }
  }
}
```

Alternatives

The current model of prescribing flat networks could be maintained with the same flat network being used for everything. This is not so much an alternative to the proposal in this spec, but rather staying with the existing solution.

Data model impact

The proposed change will be to add the following fields to the port object with their data type and default value for migrations:

Field Name	Field Type	Migration Value
local_link_connection	dict_or_none	None
portgroup_id	int_or_none	None
pxe_enabled	bool	True

All existing ports will have `pxe_enabled` set to `true` so that the current behavior is not changed. The portgroup relationship is a 1:n relationship with the port.

The portgroup object is proposed with the following fields and data types:

Field Name	Field Type
id	int
uuid	str
name	str_or_none
node_id	int_or_none
address	str_or_none
mode	str_or_none
properties	dict_or_none
extra	dict_or_none
internal_info	dict_or_none
standalone_ports_supported	bool
created_at	datetime_or_str_or_none
updated_at	datetime_or_str_or_none

Note

While `mode` attribute of the portgroup object has type str_or_none, its value can not be `None`, unless the database was changed manually. It gets populated either during the database migration, or during portgroup creation (if not specified explicitly). In both cases it is set to the `[DEFAULT]default_portgroup_mode` configuration option value.

State Machine Impact

The state machine will not be directly impacted, however, changes to the new portgroup object and additions of portgroups will only be allowed when a node is in a particular set of states.

Change to port membership of a portgroup can be made when the node is in a MANAGEABLE/INSPECTING/ENROLL state. Any port updates that update local_link_connection or pxe_enabled can only be made when the node is in a MANAGEABLE/INSPECTING/ENROLL state. The reason for limiting to these states is because updating these new port attributes should result in an update of local_link_information in the binding_profile, which would trigger an update in Neutron. It might be safest to only allow this when the node is not in a state where uninterrupted connectivity is expected. These limitations will also ensure that Neutron port updates should only happen during a state change and not automatically with any port-update call.

REST API impact

The following port API methods will be affected:

- `/v1/ports`
  - Retrieve a list of ports.
  - Method type GET.
  - The http response code(s) are unchanged. An additional reason for the 404 error http response code would be if the portgroup resource is specified but is not found.
  - New parameter can be included:
    - `portgroup (uuid_or_name)` - UUID or logical name of a portgroup to only get ports for that portgroup.

  - Body:
    - None
  - Response:
    - JSON schema definition of Port
- `/v1/ports/(port_uuid)`
  - Retrieve information about the given port.
  - Method type GET.
  - The http response code(s) are unchanged.
  - Parameter:
    - `port_uuid (uuid)` - UUID of the port.
  - Body:
    - None
  - Response:
    - JSON schema definition of Port
- `/v1/ports`
  - Create a new port.
  - Method type POST.
  - The http response code(s) are unchanged.
  - Parameter:
    - None
  - Body:
    - JSON schema definition of Port
  - Response:
    - JSON schema definition of Port
- `/v1/ports/(port_uuid)`
  - Update an existing port.
  - Method type PATCH.
  - The http response code(s) are unchanged.
  - Parameter:
    - `port_uuid (uuid)` - UUID of the port.
  - Body:
    - JSON schema definition of PortPatch
  - Response:
    - JSON schema definition of Port

- JSON schema definition of Port (data sample):

```
{
  "address": "fe:54:00:77:07:d9",
  "created_at": "2015-05-12T10:00:00.529243+00:00",
  "extra": {
    "foo": "bar",
  },
  "links": [
    {
      "href": "http://localhost:6385/v1/ports/
       1004e542-2f9f-4d9b-b8b9-5b719fa6613f",
      "rel": "self"
    },
    {
      "href": "http://localhost:6385/ports/
       1004e542-2f9f-4d9b-b8b9-5b719fa6613f",
      "rel": "bookmark"
    }
  ],
  "node_uuid": "e7a6f1e2-7176-4fe8-b8e9-ed71c77d74dd",
  "updated_at": "2015-05-15T09:04:12.011844+00:00",
  "uuid": "1004e542-2f9f-4d9b-b8b9-5b719fa6613f",
  "local_link_connection": {
    "swwitch_id": "0a:1b:2c:3d:4e:5f",
    "port_id": "Ethernet3/1",
    "switch_info": "switch1",
  },
  "portgroup_uuid": "6eb02b44-18a3-4659-8c0b-8d2802581ae4",
  "pxe_enabled": true
}
```

- JSON schema definition of PortPatch would be a subset of JSON schema of Port.

The following API methods will be added in support of the new portgroup model:

- `/v1/portgroups`
  - Retrieve a list of portgroups.
  - Method type GET.
  - Normal http response code will be 200.
  - Expected error http response code(s):
    - 400 for bad query or malformed syntax (e.g. if address is not mac-address format)
    - 404 for resource (e.g. node) not found
  - Parameters:
    - `node (uuid_or_name)` - UUID or name of a node, to only get portgroups for that node.
    - `address (macaddress)` - MAC address of a portgroup, to only get portgroup which has this MAC address.

        * `marker (uuid)` - pagination marker for large data sets.
        * `limit (int)` - maximum number of resources to return in a single result.
        * `sort_key (unicode)` - column to sort results by. Default: id.
        * `sort_dir (unicode)` - direction to sort. asc or desc. Default: asc.
    - Body:
        * None
    - Response:
        * JSON schema definition of PortgroupCollection
- `/v1/portgroups/(portgroup_ident)`
    - Retrieve information about the given portgroup.
    - Method type GET.
    - Normal http response code will be 200.
    - Expected error http response code(s):
        * 400 for bad query or malformed syntax
        * 404 for resource (e.g. portgroup) not found
    - Parameters:
        * `portgroup_ident (uuid_or_name)` - UUID or logical name of a portgroup.
    - Body:
        * None
    - Response:
        * JSON schema definition of Portgroup
- `/v1/portgroups`
    - Create a new portgroup.
    - Method type POST.
    - Normal http response code will be 201.
    - Expected error http response code(s):
        * 400 for bad query or malformed syntax
        * 409 for resource conflict (e.g. if portgroup name already exists because the name should be unique)
    - Parameters:
        * None
    - Body:
        * JSON schema definition of Portgroup
    - Response:
        * JSON schema definition of Portgroup

- `/v1/portgroups/(portgroup_ident)`
  - Delete a portgroup.
  - Method type DELETE.
  - Normal http response code will be 204.
  - Expected error http response code(s):
    - 400 for bad query or malformed syntax
    - 404 for resource (e.g. portgroup) not found
  - Parameters:
    - `portgroup_ident (uuid_or_name)` - UUID or logical name of a portgroup.
  - Body:
    - None
  - Response:
    - N/A
- `/v1/portgroups/(portgroup_ident)`
  - Update an existing portgroup.
  - Method type PATCH.
  - Normal http response code will be 200.
  - Expected error http response code(s):
    - 400 for bad query or malformed syntax
    - 404 for resource (e.g. portgroup) not found
    - 409 for resource conflict (e.g. if portgroup name already exists because the name should be unique)
  - Parameters:
    - `portgroup_ident (uuid_or_name)` - UUID or logical name of a portgroup.
  - Body:
    - JSON schema definition of PortgroupPatch
  - Response:
    - JSON schema definition of Portgroup
- `/v1/portgroups/detail`
  - Retrieve a list of portgroups with detail. The additional detail option would return all fields, whereas without it only a subset of fields would be returned, namely uuid and address.
  - Method type GET.
  - Normal http response code will be 200.
  - Expected error http response code(s):
    - 400 for bad query or malformed syntax

        * 404 for resource (e.g. node) not found
    - Parameters:
        * `node (uuid_or_name)` - UUID or name of a node, to only get portgroups for that node.
        * `address (macaddress)` - MAC address of a portgroup, to only get portgroup which has this MAC address.
        * `marker (uuid)` - pagination marker for large data sets.
        * `limit (int)` - maximum number of resources to return in a single result.
        * `sort_key (unicode)` - column to sort results by. Default: id.
        * `sort_dir (unicode)` - direction to sort. asc or desc. Default: asc.
    - Body:
        * None
    - Response:
        * JSON schema definition of PortgroupCollection with detail.
- `/v1/nodes/(node_ident)/portgroups`
    - Retrieve a list of portgroups for node.
    - Method type GET.
    - Normal http response code will be 200.
    - Expected error http response code(s):
        * 400 for bad query or malformed syntax
        * 404 for resource (e.g. node) not found
    - Parameters:
        * `node_ident (uuid_or_name)` - UUID or logical name of a node.
    - Body:
        * None
    - Response:
        * JSON schema definition of PortgroupCollection.
- `/v1/nodes/(node_ident)/portgroups/detail`
    - Retrieve a list of portgroups with detail for node.
    - Method type GET.
    - Normal http response code will be 200.
    - Expected error http response code(s):
        * 400 for bad query or malformed syntax
        * 404 for resource (e.g. node) not found
    - Parameters:
        * `node_ident (uuid_or_name)` - UUID or logical name of a node.

  - Body:
    - None
  - Response:
    - JSON schema definition of PortgroupCollection with detail.
- `/v1/portgroups/(portgroup_ident)/ports`
  - Retrieve a list of ports for portgroup.
  - Method type GET.
  - Normal http response code will be 200.
  - Expected error http response code(s):
    - 400 for bad query or malformed syntax
    - 404 for resource (e.g. portgroup) not found
  - Parameters:
    - `portgroup_ident (uuid_or_name)` - UUID or logical name of a portgroup.
  - Body:
    - None
  - Response:
    - JSON schema definition of PortCollection.
- `/v1/portgroups/(portgroup_ident)/ports/detail`
  - Retrieve a list of ports with detail for portgroup.
  - Method type GET.
  - Normal http response code will be 200.
  - Expected error http response code(s):
    - 400 for bad query or malformed syntax
    - 404 for resource (e.g. portgroup) not found
  - Parameters:
    - `portgroup_ident (uuid_or_name)` - UUID or logical name of a portgroup.
  - Body:
    - None
  - Response:
    - JSON schema definition of PortCollection with detail.
- JSON schema definition of Portgroup (data sample):

```
{
  "address": "fe:54:00:77:07:d9",
  "created_at": "2015-05-12T10:10:00.529243+00:00",
  "extra": {
```

(continues on next page)

(continued from previous page)


```
    "foo": "bar",
  },
  "internal_info": {},
  "links": [
    {
      "href": "http://localhost:6385/v1/portgroups/
      6eb02b44-18a3-4659-8c0b-8d2802581ae4",
      "rel": "self"
    },
    {
      "href": "http://localhost:6385/portgroups/
      6eb02b44-18a3-4659-8c0b-8d2802581ae4",
      "rel": "bookmark"
    }
  ],
  "mode": "802.3ad",
  "name": "node1_portgroup1",
  "node_uuid": "e7a6f1e2-7176-4fe8-b8e9-ed71c77d74dd",
  "ports": [
    {
      "href": "http://127.0.0.1:6385/v1/portgroups/
      6eb02b44-18a3-4659-8c0b-8d2802581ae4/ports",
      "rel": "self"
    },
    {
      "href": "http://127.0.0.1:6385/portgroups/
      6eb02b44-18a3-4659-8c0b-8d2802581ae4/ports",
      "rel": "bookmark"
    }
  ],
  "properties": {
    "bond_xmit_hash_policy": "layer3+4",
    "bond_miimon": 100
  },
  "standalone_ports_supported": true,
  "updated_at": "2015-05-15T09:04:12.011844+00:00",
  "uuid": "6eb02b44-18a3-4659-8c0b-8d2802581ae4"
}
```


- JSON schema definition of PortgroupCollection:


```
{
  "portgroups": [
      {
          "address": "fe:54:00:77:07:d9",
          "links": [
              {
                  "href": "http://localhost:6385/v1/portgroups/
                  6eb02b44-18a3-4659-8c0b-8d2802581ae4",
                  "rel": "self"
```


(continues on next page)

(continued from previous page)

```
            },
            {
                "href": "http://localhost:6385/portgroups/
                6eb02b44-18a3-4659-8c0b-8d2802581ae4",
                "rel": "bookmark"
            }
        ],
        "name": "node1_portgroup1",
        "uuid": "6eb02b44-18a3-4659-8c0b-8d2802581ae4"
    }
  ]
}
```

- JSON schema definition of PortgroupCollection with detail:

```
{
  "portgroups": [
    {
      "address": "fe:54:00:77:07:d9",
      "created_at": "2016-08-18T22:28:48.165105+00:00",
      "extra": {},
      "internal_info": {},
      "links": [
        {
          "href": "http://127.0.0.1:6385/v1/portgroups/
          6eb02b44-18a3-4659-8c0b-8d2802581ae4",
          "rel": "self"
        },
        {
          "href": "http://127.0.0.1:6385/portgroups/
          6eb02b44-18a3-4659-8c0b-8d2802581ae4",
          "rel": "bookmark"
        }
      ],
      "mode": "802.3ad",
      "name": "node1_portgroup1",
      "node_uuid": "e7a6f1e2-7176-4fe8-b8e9-ed71c77d74dd",
      "ports": [
        {
          "href": "http://127.0.0.1:6385/v1/portgroups/
          6eb02b44-18a3-4659-8c0b-8d2802581ae4/ports",
          "rel": "self"
        },
        {
          "href": "http://127.0.0.1:6385/portgroups/
          6eb02b44-18a3-4659-8c0b-8d2802581ae4/ports",
          "rel": "bookmark"
        }
      ],
      "properties": {
```

(continues on next page)

(continued from previous page)

```
        "bond_xmit_hash_policy": "layer3+4",
        "bond_miimon": 100
      },
      "standalone_ports_supported": true,
      "updated_at": "2016-11-04T17:46:09+00:00",
      "uuid": "6eb02b44-18a3-4659-8c0b-8d2802581ae4"
    }
  ]
}
```

- JSON schema definition of PortgroupPatch would be a subset of JSON schema of Portgroup.

Does the API microversion need to increment?

- Yes.

Example use case including typical API samples for both data supplied by the caller and the response.

- Example of port create.
  - Data supplied:

```
{
  "address": "fe:54:00:77:07:d9",
  "node_uuid": "e7a6f1e2-7176-4fe8-b8e9-ed71c77d74dd",
  "local_link_connection": {
    "switch_id": "0a:1b:2c:3d:4e:5f",
    "port_id": "Ethernet3/1",
    "switch_info": "switch1",
    },
  "pxe_enabled": true
}
```

  - Response 201 with body:

```
{
  "address": "fe:54:00:77:07:d9",
  "node_uuid": "e7a6f1e2-7176-4fe8-b8e9-ed71c77d74dd",
  "local_link_connection": {
    "switch_id": "0a:1b:2c:3d:4e:5f",
    "port_id": "Ethernet3/1",
    "switch_info": "switch1",
    },
  "pxe_enabled": true
  "created_at": "2015-05-12T10:00:00.529243+00:00",
  "extra": {
  },
  "links": [
    {
      "href": "http://localhost:6385/v1/ports/
       1004e542-2f9f-4d9b-b8b9-5b719fa6613f",
      "rel": "self"
```

(continues on next page)

(continued from previous page)


```
    },
    {
      "href": "http://localhost:6385/ports/
       1004e542-2f9f-4d9b-b8b9-5b719fa6613f",
      "rel": "bookmark"
    }
  ],
  "updated_at": null,
  "uuid": "1004e542-2f9f-4d9b-b8b9-5b719fa6613f",
  "portgroup_uuid": null,
}
```


- Example of portgroup create.
  - Data supplied:


```
{
  "address": "fe:54:00:77:07:d9",
  "node_uuid": "e7a6f1e2-7176-4fe8-b8e9-ed71c77d74dd",
  "standalone_ports_supported": true,
  "name": "node1_portgroup1"
}
```


  - Response 201 with body:


```
{
  "address": "fe:54:00:77:07:d9",
  "node_uuid": "e7a6f1e2-7176-4fe8-b8e9-ed71c77d74dd",
  "name": "node1_portgroup1"
  "created_at": "2015-05-12T10:10:00.529243+00:00",
  "extra": {
  },
  "internal_info": {},
  "links": [
    {
      "href": "http://localhost:6385/v1/portgroups/
       6eb02b44-18a3-4659-8c0b-8d2802581ae4",
      "rel": "self"
    },
    {
      "href": "http://localhost:6385/portgroups/
       6eb02b44-18a3-4659-8c0b-8d2802581ae4",
      "rel": "bookmark"
    }
  ],
  "mode": null,
  "ports": [
    {
      "href": "http://127.0.0.1:6385/v1/portgroups/
      6eb02b44-18a3-4659-8c0b-8d2802581ae4/ports",
```


(continues on next page)

(continued from previous page)


```
      "rel": "self"
    },
    {
      "href": "http://127.0.0.1:6385/portgroups/
      6eb02b44-18a3-4659-8c0b-8d2802581ae4/ports",
      "rel": "bookmark"
    }
  ],
  "properties": {},
  "standalone_ports_supported": true,
  "updated_at": null,
  "uuid": "6eb02b44-18a3-4659-8c0b-8d2802581ae4",
}
```


- Example of port update.
  - Parameter port_uuid=1004e542-2f9f-4d9b-b8b9-5b719fa6613f
  - Data supplied (JSON PATCH syntax where op can be add/replace/delete):


```
[{"path": "/portgroup_uuid", "value":
  "6eb02b44-18a3-4659-8c0b-8d2802581ae4", "op": "add"}]
```


  - Response 200 with body:


```
{
  "address": "fe:54:00:77:07:d9",
  "node_uuid": "e7a6f1e2-7176-4fe8-b8e9-ed71c77d74dd",
  "local_link_connection": {
    "switch_id": "0a:1b:2c:3d:4e:5f",
    "port_id": "Ethernet3/1",
    "switch_info": "switch1",
    },
  "pxe_enabled": true
  "created_at": "2015-05-12T10:00:00.529243+00:00",
  "extra": {
  },
  "links": [
    {
      "href": "http://localhost:6385/v1/ports/
       1004e542-2f9f-4d9b-b8b9-5b719fa6613f",
      "rel": "self"
    },
    {
      "href": "http://localhost:6385/ports/
       1004e542-2f9f-4d9b-b8b9-5b719fa6613f",
      "rel": "bookmark"
    }
  ],
  "updated_at": "2015-05-12T10:20:00.529243+00:00",
  "uuid": "1004e542-2f9f-4d9b-b8b9-5b719fa6613f",
```


(continues on next page)

(continued from previous page)

```
  "portgroup_uuid": "6eb02b44-18a3-4659-8c0b-8d2802581ae4",
}
```

  - Note that the port update API should support updating the portgroup_id of the port object. This will allow operators to migrate existing deployments.

- Example of port list.
  - Parameter node_uuid=e7a6f1e2-7176-4fe8-b8e9-ed71c77d74dd
  - Response 200 with body:

```
{"ports": [
  {
  "address": "fe:54:00:77:07:d9",
  "links": [
    {
      "href": "http://localhost:6385/v1/ports/
       1004e542-2f9f-4d9b-b8b9-5b719fa6613f",
      "rel": "self"
    },
    {
      "href": "http://localhost:6385/ports/
       1004e542-2f9f-4d9b-b8b9-5b719fa6613f",
      "rel": "bookmark"
    }
  ],
  "uuid": "1004e542-2f9f-4d9b-b8b9-5b719fa6613f",
  "portgroup_uuid": "6eb02b44-18a3-4659-8c0b-8d2802581ae4",
  }
]}
```

  - Note that portgroup_uuid is now returned in the response.

Discuss any policy changes, and discuss what things a deployer needs to think about when defining their policy.

- Ironic has an admin-only policy so policy definitions should not be a concern.
- A deployer should be aware of the capabilities of the particular ML2 driver for supporting use of the new local_link_information that will be passed to it via the binding_profile.

Is a corresponding change in the client library and CLI necessary?

- The client library and CLI should be updated to support the new APIs.

Is this change discoverable by clients? Not all clients will upgrade at the same time, so this change must work with older clients without breaking them.

- The changes to the API will be backward-compatible so older clients will still continue to work as-is.

Client (CLI) impact

The python-ironicclient and OSC would need updated to support the new portgroups APIs.

In the commands below, <portgroup> means that this placeholder can contain both portgroup UUID or name. <portgroup_uuid> can contain only portgroup UUID.

Example usage of the new methods:

- For ports, the CLI would support port creation with new optional parameters specifying the new port attributes (local_link_connection, portgroup_id and pxe_enabled) and would also support update of these attributes. As examples:

  ironic CLI:

  - ironic port-create -a <address> -n <node> [-e <key=value>] [local-link-connection <key=value>] [portgroup <portgroup>] [pxe-enabled <boolean>]
  - ironic port-update <port_uuid> add portgroup_uuid=<portgroup_uuid> local-link-connection <key=value> pxe-enabled <boolean>

  openstack baremetal CLI:

  - openstack baremetal port create node <node> [local-link-connection <key=value>] [port-group <portgroup>] [pxe-enabled <boolean>] <address>
  - openstack baremetal port set [port-group <portgroup>] [local-link-connection <key=value>] [pxe-enabled <boolean>] <port>
  - openstack baremetal port list [address <mac-address>] [node <node> | port-group <port-group>]

- For portgroups, the CLI would support the following new methods:

  ironic CLI:

  - ironic portgroup-create node <node> [address <mac-address>] [name <portgroupname>] [-e <key=value>] [standalone-ports-supported <boolean>] [-m <mode>] [-p <key=value>]
  - ironic portgroup-delete <portgroup> [<portgroup>]
  - ironic portgroup-list [detail | fields <field> [<field>]] [node <node>] [address <mac-address>] [limit <limit>] [marker <portgroup_uuid>] [sort-key <field>] [sort-dir <direction>]
  - ironic portgroup-port-list [detail | fields <field> [<field>]] [limit <limit>] [marker <port-group_uuid>] [sort-key <field>] [sort-dir <direction>] <portgroup>
  - ironic portgroup-show [address] [fields <field> [<field>]] <id>
    - <id> is the UUID or name of the portgroup (or MAC address if address is specified)
  - ironic portgroup-update <portgroup> <op> <path=value> [<path=value>]
    - <op> is add, remove or replace.
    - <path=value> is the attribute to add, remove or replace. Can be specified multiple times. For remove only <path> is necessary.
  - Note: Even though the ironic CLI includes ironic node-port-list, we are NOT going to provide a corresponding ironic node-portgroup-list. Rather, the list of portgroups of a node will be available via ironic portgroup-list node.

openstack baremetal CLI:

- openstack baremetal port group create node <uuid> [name <name>] [extra <key=value>] [support-standalone-ports | unsupport-standalone-ports] [mode <mode>] [properties <key=value>] [address <mac-address>]
- openstack baremetal port group delete <portgroup> [<portgroup>]
- openstack baremetal port group list [marker <portgroup>] [address <mac-address>] [node <node>] [sort <key>[:<direction>]] [long | fields <field> [<field>]]
- openstack baremetal port group show [address] [fields <field> [<field>]] <id>
  - <id> is the UUID or name of the portgroup (or MAC address if address is specified)
- openstack baremetal port group set [address <mac-address>] [name <name>] [node <node>] [extra <key=value>] [support-standalone-ports | unsupport-standalone-ports] [mode <mode>] [properties <key=value>] <portgroup>
- openstack baremetal port group unset [address] [name] [extra <key>] [properties key] <portgroup>
- To add ports to a portgroup, the portgroup should first be created and then port_update or port create called.

The python-ironicclient would also need the Port detailed resource extended to include the new port attributes.

RPC API impact

No impact on existing API calls.

New RPC API calls would be needed:

- update_portgroup
- destroy_portgroup

These new API calls will use call(). As for the existing API call for update_port, the new API call for update_portgroup should request an update for DHCP if the address field is updated.

To roll this change out to an existing deployment, the ironic-conductor should be upgraded before the ironic-api.

Driver API impact

The NeutronDHCPApi class in `ironic/dhcp/neutron` updates Neutron ports with DHCP options. The vifs are obtained in `ironic/common/network` by extracting `vif_port_id` from the `extra` attributes of Ironic ports. This method should be updated if vifs are bound to portgroups as well as ports.

The complementary network-provider spec[Page 522, 1] provides details regarding the workflow of the network flip and the point at which the binding profile will be passed to Neutron to bind the port.

Nova driver impact

As this work depends on the attach/detach interface work[2], the only thing that needs to be changed to fully support portgroups is configdrive generation.

[2] http://specs.openstack.org/openstack/ironic-specs/specs/approved/interface-attach-detach-api.html

Nova will call into ironic to get the list of ports of each portgroup that has a VIF associated with it, along with portgroup `mode` and `properties` fields (see *Data model impact* section), and update the network metadata with the needed information. When the contents of the `properties` dictionary gets passed to the config drive builder in nova, we will ensure that `bond_` prefix is prepended to all key names, so that these keys are not ignored by cloud-init when reading the config drive.

Ramdisk impact

N/A

Security impact

The new REST API calls for portgroups should not be usable by the end user. Only operators and administrators should be able to manage portgroups and local_link_connection data of ports, because these settings are used to configure the network. This is satisfied because Ironic is an admin-only API, so there should be no security impact.

Other end user impact

Using the binding profile to enable flipping between provisioning and tenant networks means there will be no support for PXE booting after deploy (i.e. local disk installation only). How to allow operators to deploy instances using either net-boot or local boot using the same Ironic conductor should be discussed in the complementary network-provider spec[Page 522, 1].

Scalability impact

There will be more API calls made to Ironic in order to create and use portgroups but impact on scalability should be negligible.

Performance Impact

None.

Other deployer impact

New database columns are added to the port table and a new database table portgroup is introduced, so this will require a database migration.

Deployers will need to deploy an ML2 mechanism driver that supports connecting baremetal resources to Neutron networks.

If using Nova, deployers will need to deploy a version of Nova that supports this feature.

Deployers may want to set the `[DEFAULT]default_portgroup_mode` configuration option to match their environment. Its default value is `active-backup`.

Deployers should be aware that automated upgrade or migration for already-provisioned nodes is not supported. Deployers should follow this recommendation for upgrading a node in an existing deployment to use this new feature:

- Upgrade the OpenStack services.
- Move node into the MANAGEABLE state.
- Update node driver field (see the network-provider spec[Page 522, 1]).

- Create Ironic portgroups.
- Update Ironic port membership to portgroups.
- Update Ironic ports with local_link_connection data.
- Move node into the AVAILABLE state.

Developer impact

Neutron ML2 mechanism drivers should support this feature by using the data passed in binding profile to dynamically configure relevant ports and port-channels on the relevant switch(es).

Implementation

Assignee(s)

- laura-moore
- yhvh (Will Stevenson)
- bertiefulton
- sukhdev-8
- vsaienko
- vdrok

Work Items

- Extend port table.
- Create the new portgroup table.
- Implement extension to port APIs.
- Implement the new portgroup APIs.
- Implement the extension to the RPC API.
- Implement the changes to the Nova driver to get and use the binding profile.
- Implement the changes needed to get vifs for updating Neutron port DHCP options.
- Implement tests for the new functionality.
- Implement updates to the python-ironicclient.
- Update documentation.

Dependencies

Network flip is dependent on the network-provider spec[Page 522, 1].

Nova changes are dependent on attach/detach interfaces work[Page 539, 2].

VLAN provisioning on switch(es) is dependent on ML2 driver functionality being developed to support this feature.

Testing

Existing default behaviour will be tested in the gate by default.

New tests will need to be written to test the new APIs and database updates.

Simulation of connecting real hardware to real switches for testing purposes is described in the network-provider spec[Page 522, 1].

Upgrades and Backwards Compatibility

Default behavior is the current behavior, so this change should be fully backwards compatible.

Documentation Impact

This feature will be fully documented.

References

Discussions on the topic include:

- https://etherpad.openstack.org/p/YVR-neutron-ironic
- https://etherpad.openstack.org/p/liberty-ironic-network-isolation
- https://etherpad.openstack.org/p/network-interface-vifs-configdrive
- Logs from https://wiki.openstack.org/wiki/Meetings/Ironic-neutron

Add notification support to ironic

https://bugs.launchpad.net/ironic/+bug/1526408

Near real-time inter-service notifications in OpenStack have a variety of use cases. At the Mitaka design summit, some of the use cases discussed for ironic included integration with billing systems, using notifications to support an operations panel to keep track of potential failures, and integration with TripleO.[1] Nova has supported some form of notifications since the Diablo cycle.[2] This spec proposes a design for a general notification schema thats versioned and structured but extensible enough to support future needs.

Problem description

Ironic doesnt currently support any type of notifications for external services to consume.

This makes it difficult for operators to monitor and keep track of events in ironic like node state changes, node power events, and successful or failed deployments. Sending notifications that contain relevant information about a node at different points in time, like state transitions, would make it easier for operators to use external tools to debug unexpected behavior.

Not having notifications also makes it difficult for external services to identify state changes in ironic. Nova and inspector have to poll ironic repeatedly to see when a state change has occurred. Adding notifications for cases like these will allow ironic to push out these event changes rather than putting the bulk of this burden on external services.

This is especially problematic if ironic were ever to be used in a standalone deployment without an external service like Nova that keeps track of resources and emits its own notifications, since the deployers

[1] Summit discussion: https://etherpad.openstack.org/p/summit-mitaka-ironic-notifications-bus
[2] https://blueprints.launchpad.net/nova/+spec/notification-system

of ironic wouldnt have any way to keep track of any state changes outside of querying the ironic API periodically or looking at logs.

Proposed change

Notification support will be added to ironic using the notifier interface from oslo.messaging.[3] oslo.messaging uses the following format for notifications:

```
{'message_id': six.text_type(uuid.uuid4()),
 'publisher_id': 'compute.host1',
 'timestamp': timeutils.utcnow(),
 'priority': 'WARN',
 'event_type': 'compute.create_instance',
 'payload': {'instance_id': 12, ... }}
```

Ironic notifications will adhere to that format.

Base classes will be created to model the notification format. The fields of the notification that arent auto-generated (everything except message_id and timestamp) will be defined as follows for ironic:

- publisher_id will be taken from the service emitting the notification and the hostname
- priority will be one of DEBUG, WARN, INFO, or ERROR, depending on the notification. Whichever level a notification uses should follow the OpenStack logging standards.[4]
- event_type will be a short string describing the notification. Each string will start with baremetal. to distinguish ironic notifications from other notifications on the message bus.

  This will be followed by the object thats being acted on, a descriptor of the action being taken, and the status of the action (start, end, error, or success). These statuses will have the following semantics:

  - start will be used when an action that is not immediate begins.
  - end will be used when an action that is not immediate succeeds.
  - error will be used when an action fails, regardless of whether the action is immediate or not.
  - success will be used when an action that succeeds almost immediately, and there is no need to report the intermediate resource states.

  event_type will have a base class defining these fields, with subclasses for each type of event that occurs. These subclasses will be versioned objects. Each of these fields will be separated by a period in the event_type string sent with the notification.

  Given the above information, event_type will end up as a string on the message bus with the following format: `baremetal.<object>.<action>.<status>`

  A few examples of potential event_type strings:

  - baremetal.node.power_set.start
  - baremetal.node.power_set.end
  - baremetal.node.provision_set.start
  - baremetal.node.provision_set.end

[3] http://docs.openstack.org/developer/oslo.messaging/notifier.html
[4] https://wiki.openstack.org/wiki/LoggingStandards#Log_level_definitions

  - baremetal.conductor.init.success
  - baremetal.conductor.stop.success

- payload will be a versioned object related to the notification topic. This should include relevant information about the event being notified on. For the notifications above about a node, for example, we would send an object containing a limited subset of the fields of the Node object, removing sensitive fields like private credentials as well as fields with an unknown length, along with the version of the notification object. A separate versioned notification object must be created for each type of notification to keep notification versions separate from other object versions in ironic. This is to prevent notifications from automatically changing as new fields are added to other objects like the Node object.

The initial implementation will consist of the base classes needed for modeling the notification format and emitting notifications as well as the power state notifications described above.

Alternatives

One alternative would be to modify oslo.messagings notifier interface to provide the base class defining a notification as a versioned object. The advantage of this is that we wouldnt have to rely on the payload to do versioning. The disadvantage is that it would require significant changes to oslo.messaging which affect other projects and require changes to every notification consumer as well as any downstream tooling built around the existing notification format.

We could also have a global notification version that gets incremented with each change to any notification. This doesnt seem desirable because it could potentially miss changes if a separate ironic object included in the payload like the Node object gets changed and the notification itself doesnt get its version bumped.

Data model impact

No database changes will be required for this feature.

The change will introduce new versioned base objects for the general notification schema and additional subclasses for individual notification types. The base classes will look similar to those in the proposal in the nova versioned notifications spec.[5]

State Machine Impact

No impact on states or transitions themselves.

Notifications will be sent out on node provision state changes if notifications are enabled. This can be achieved by sending notifications every time a TaskManagers process_event method starts and successfully executes.

All information related to a nodes previous, current, and new target state will be included in the notifications. The .start notification will have the current provision_state before the state change and the target_provision_state. After an event is successfully processed, the .end notification will include the current provision_state (the .start notifications target state) and the new target_provision_state. The notifications will also include the name of the event that caused the state change. This will be useful for disambiguating between cases where there are multiple potential transitions from one state to another.

[5] Nova versioned notifications spec: https://github.com/openstack/nova-specs/blob/master/specs/mitaka/approved/versioned-notification-api.rst

REST API impact

None.

Client (CLI) impact

None.

RPC API impact

No impact from an API standpoint.

Modifications to the implementation of certain conductor RPC API methods will need to be made for notifications that are sent when an RPC is dispatched to a worker, however. See Driver API Impact for an example of how this might be done for power notifications.

Driver API impact

No impact from an API standpoint.

Notifications related to power state changes will be added, but that can be done without modifying any of the driver classes in the following manner:

1) Send a baremetal.node.set_power_state.start notification after the ConductorManager receives the change_node_power_state call as a conductor background task.

2) On success, after the dispatched call to node_power_action finishes without raising an exception, send a baremetal.node.set_power_state.end notification.

3) On error, the power_state_error_handler hook will be called in the conductor manager. Send a baremetal.node.set_power_state.error notification here.

Nova driver impact

None.

Ramdisk impact

N/A

Security impact

None.

Other end user impact

None, except a message bus will have to be used if a deployer wants to use the notification system.

Scalability impact

When enabled, notifications will put additional load on whichever message bus the notifications are sent to.

Performance Impact

When enabled, code to send the notification will be called each time an event occurs that triggers a notification. This shouldnt be much of a problem for ironic itself, but the load on whatever message bus is used should be considered (see Scalability Impact).

Other deployer impact

The following configuration options will be added:

- The notification_transport_url option needed by oslo.messaging.[6] Defaults to None which indicates that the same configuration thats used for RPC will be used.
- A notification_level string parameter will be added to indicate the minimum priority level for which notifications will be sent. Available options will be DEBUG, INFO, WARN, ERROR, or None to disable notifications. None will be the default.

  An alternative to the notification_level global config option would be to create specific config options defining whether a particular notification type should be sent. This is what nova does, but summit discussions indicated that consistency is preferable.

Developer impact

Developers should adhere to proper versioning guidelines and use the notification base classes when creating new notifications.

Implementation

Assignee(s)

Primary assignee:

- mariojv

Other contributors:

- lucasagomes

Work Items

- Create notification base classes and tests
- Write documentation for how to use the base classes consistently across all ironic notifications
- Implement an example of a notification for when a node power state is changed

Dependencies

None.

[6] http://docs.openstack.org/developer/oslo.messaging/opts.html

Testing

Unit tests for both the base classes and the node power state notification will be added.

Upgrades and Backwards Compatibility

No impact, but modifications to notifications created in the future must be checked for backwards compatibility.

Documentation Impact

- Developer documentation will be added for how to add new notifications or modify existing notifications
- Document an example of what an emitted notification will look like
- Documentation should be added for each notification indicating when its expected to be emitted

References

5.11 Newton

5.11.1 6.2

Add InfiniBand Support

https://bugs.launchpad.net/ironic/+bug/1532534

Today, Ironic supports Ethernet interfaces for Hardware inspection and PXE boot. Ironic should have the ability to inspect and PXE boot over InfiniBand network as well.

Problem description

- Hardware inspection for InfiniBand - A InfiniBand GUID is similar in concept to a MAC address because it consists of a 24-bit manufacturers prefix and a 40-bit device identifier (64 bits total).
- PXE Boot over InfiniBand - To allow DHCP over IPoIB interface the DHCP client must send the client-id with a unique identifying value for the client. The value is consist of the vendor prefix 12 bytes and GUID 8 bytes. Today ironic doesnt send update the neutron port with client-id option.

Proposed change

- Hardware inspection for InfiniBand - In order to use InfiniBand with PXE you have to flash the NIC with a vendor specific firmware. The vendor firmware defined the conversion of GUID to InfiniBand MAC (48 bits). To resource complicity of the change, the ironic address will contain the InfiniBand MAC.
- PXE/iPXE Boot over InfiniBand changes: To allow DHCP over InfiniBand we need the following:
  1. The dhcp-server must use the BROADCAST flag in the dhcp-server. This already support in neutron-dhcp-agent by config file.
  2. Updating the ironic port extra attribute to contains the InfiniBand port client-id extra e.g:

```
{
    'client-id':
    'ff:00:00:00:00:00:02:00:00:02:c9:00:00:02:c9:03:00:00:10:39'
}
```

The client-id update can be done manually or with IPA and ironic-inspector.

3. The neutron port that represent the ironic port should be updated with client-id option in the extra_dhcp_opts attribute. The client-id consists of a vendor prefix and the port GUID. The client id for Mellanox ConnectX Family Devices is consists of a prefix (ff:00:00:00:00:00:02:00:00:02:c9:00) and 8 byte port GUID. The prefix in the client-id is vendor specific.
4. The PXE MAC file name consists of the <Hardware Type>-<MAC>. For InfiniBand the hardware Type is 20 and the mac is the InfiniBand truncate GUID.
5. The iPXE MAC file name consists of the <MAC>. For InfiniBand the MAC is the InfiniBand truncate GUID.

Other projects changes:

- ironic-python-agent changes:
  1. Update the ironic agent to calculate the InfiniBand truncate GUID and the Client ID.
  2. Update coreos and tinyipa with ib_ipoib driver.
- ironic-inspector changes:
  1. Update the ironic-inspector to update port.extra with client-id
- diskimage-builder changes:
  1. Update the mellanox element to load ib_ipoib driver.

Alternatives

- Extend the ironic port to support GUID which is 8 bytes and calculate the client-id in the ironic code from the GUID. This will require updating the ironic model and API. This will require updating the nova ironic driver to truncate the GUID to MAC.

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

ironic CLI

None

openstack baremetal CLI

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

N/A

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

When using IPA, the deployer needs IPA that provides the InfiniBand MAC and client-id.

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
moshele

Other contributors:
None

Work Items

- Add Client-ID option to the neutron port to allow DHCP.
- Update the generation of the iPXE/PXE file.
- Update documentation.

Dependencies

None

Testing

- Adding unit tests.
- Adding Third-party CI which will test Mellanox hardware.

Upgrades and Backwards Compatibility

None

Documentation Impact

- We will update the ironic documentation on how to allow pxe boot from IPoIB.

References

- http://www.syslinux.org/wiki/index.php/PXELINUX
- https://tools.ietf.org/html/rfc4392
- http://www.mellanox.com/related-docs/prod_software/Mellanox_FlexBoot_User_Manual_v2.3.pdf

Collect logs from agent ramdisk

https://bugs.launchpad.net/ironic/+bug/1587143

This spec adds support for retrieving the deployment system logs from agent ramdisk.

Problem description

Currently we do not have any documented mechanism to automatically retrieve the system logs from Bootstrap. Having access to agent ramdisk logs may be highly required especially after deployment failure. Collecting logs on remote server is common deployment thing. There are a lot of proprietary, opensource, standardized technologies that solve this task. Lets walk through linux natively supported:

1. Syslog: is the standardized protocol for message logging, described in RFC. All distros supports syslog in different implementations `rsyslogd`, `syslogd`, `syslog-ng`
2. Systemd journal-remote: systemd tool to send/receive journal messages over the network.

Proposed change

The proposed implementations is about adding documentation that helps Operators to build agent ramdisk with collecting logs feature. On top of syslog and systemd journal-remote technologies.

Changes in IPA

A new kernel parameter is added `use_syslog`.

1. `use_syslog` (Boolean): Whether IPA should send the logs to syslog or not. Defaults to False.

Changes in Ironic

New pxe_append parameters are added `use_syslog` and `syslog_server` in `[agent]` section.

1. `use_syslog` boolean): Whether IPA should send the logs to syslog or not. Defaults to False.
2. `syslog_server` (string): IP address or DNS name of syslog server to send system logs to. Defaults to IP address of ironic-conductor node. The variable may be used to configure `rsyslogd` or `syslog` or any other syslog server implementation.

Changes in agent ramdisk

Bootstrap should support remote logging via syslog/systemd. All agent ramdisk changes are related to logging tool configuration (rsyslogd, syslogd, journal-remote). The logging service should be started right after networking service to make sure that we send all logs from OpenStack services. On `Collector` side (ironic-conductor) we receive messages from `Originator` (agent ramdisk) and place the according to local rules. It may be directory on the filesystem: /var/log/remote/ironic/<node_uuid>/<log_filename>

Alternatives

Implement logs collecting mechanism in Ironic and IPA as it described in https://review.opendev.org/323511

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

Support will be added to ramdisk build tools, such as `diskimage-builder` and image building scripts under IPA repository to build ramdisks with remote logging support.

Security impact

None.

Other end user impact

None.

Scalability impact

None

Performance Impact

None

Other deployer impact

Deployer chooses which mechanism for collecting logs should be used. General changes to agent ramdisk described at *Changes in agent ramdisk*

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
lucasagomes <lucasagomes@gmail.com> vsaienko <vsaienko@mirantis.com>

Other contributors:

Work Items

- Add documentation that describes how to configure `Collector` on ironic-conductor, and `Originator` in ironic agent ramdisk on top of rsyslogd or syslogd or journal-remote.
- Add an examples of building most popular images (coreos, tinyipa) with syslog support. New `dib` element will be added as well.
- Integrate configuring `Collector` on devstack host.

Dependencies

None

Testing

Collecting logs from agent ramdisk will be configured on gate jobs. The directory with logs from agent ramdisk will be archived and available in the jobs artefacts.

Upgrades and Backwards Compatibility

None.

Documentation Impact

Documentation will be provided about how to configure syslog `Collector` to receive and store logs from `Originator`.

References

None.

iPXE to use Swift Temporary URLs

https://bugs.launchpad.net/ironic/+bug/1526404

This adds support for generating Swift temporary URLs for the deploy ramdisk and kernel when booting with iPXE.

Problem description

Currently the iPXE driver requires an external HTTP server to serve the deploy ramdisk and kernel. When used with Glance, the `ironic-conductor` fetches the images from it and place them under the HTTP root directory, and if a rebalance happens in the hash right the new `ironic-conductor` taking over the node have to do the same thing, fetch the images and cache it locally to be able to manage that node.

Having an external HTTP server should not be required when Glance is used with a Swift backend, with Swift we can generate temporary URLs that can be passed to iPXE to download the images without requiring credentials.

Proposed change

The proposed implementation consists in having the iPXE driver to create a Swift tempurl[0] for the deploy ramdisk and kernel that the node will boot as part of the config generation.

This also proposes adding a boolean configuration option under the `pxe` group called `ipxe_use_swift`. If True this will tell iPXE to not cache the images in the disk and generate the Swift tempurl for the ramdisk and kernel, if False, iPXE will continue to cache the images under the HTTP root directory. Defaults to False.

Note that in order to keep compatibility with Nova behavior, kernel/ramdisk of the user image still have to be cached in case `netboot` is required. Doing otherwise will make it impossible for user to reboot the instance from within when tempurls have expired or the image is deleted from Glance altogether.

Alternatives

Continue to use an external HTTP server and caching the images on the disk.

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

RPC API impact

None

[0] http://docs.openstack.org/kilo/config-reference/content/object-storage-tempurl.html

Driver API impact

None

Nova driver impact

None

Ramdisk impact

N/A

Security impact

Theres a positive security impact because the Swift temporary URLs does have an expiration time and the images in the external HTTP server will be available until the instance is destroyed.

Other end user impact

None

Scalability impact

There is a scaling benefit to download directly from Swift since a Swift cluster can be scaled horizontally by adding new nodes.

Performance Impact

None

Other deployer impact

None

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
lucasagomes <lucasagomes@gmail.com>

Other contributors:
pshchelo <shchelokovskyy@gmail.com>

Work Items

- Add the new `ipxe_use_swift` configuration option under the `pxe` group.
- Get the PXE driver to generate the Swift temporary URLs as part of the configuration generation when `ipxe_use_swift` is True.

- Skip caching the image on the disk when `ipxe_use_swift` is True.

Dependencies

None

Testing

Unittests will be added.

Upgrades and Backwards Compatibility

None

Documentation Impact

The iPXE documentation will be updated to reflect the changes made by this spec.

References

Implementation of ironic commands as an OSC plugin

https://bugs.launchpad.net/ironic/+bug/1526479

The OpenStackClient is becoming the defacto CLI client for OpenStack. This spec will spell out what the command structure should look like, including parameters and command names.

Problem description

The OpenStackClient[1] has become the preferred method of creating clients for OpenStack APIs. The initial implementation has been done. What needs to happen is to define what the command structure should be. There has been some confusion/discussion[2] about what these commands should look like, so it seemed the proper thing to create a spec.

The goal of the OpenStackClient is to make the CLI clients more natural for the End User. This spec will specify the commands that the End User will use when interacting with Ironic.

Proposed change

The proposed implementation will have all of the commands implemented as specified in the *Client (CLI) impact* section below.

In addition (or as clarification) to the OpenStackClient command structure[3] :

- the OpenStackClient command structure is described as `<object1> <action> <object2>`. This doesnt work if there are commands of the form `<object1> <action>`. Instead, we will use the form `<object1> <object2> <action>`. (Perhaps think of it as one object with two parts). For example, instead of openstack baremetal node set maintenance (because we have openstack baremetal node set), we will use openstack baremetal node maintenance set.

[1] http://docs.openstack.org/developer/python-openstackclient/index.html
[2] http://lists.openstack.org/pipermail/openstack-dev/2015-November/078998.html
[3] http://docs.openstack.org/developer/python-openstackclient/commands.html

- dont use hyphenated nouns, because the commands should be more natural and there arent any commands (yet) that use hyphens. For example, instead of openstack baremetal node boot-device set, we are going to use openstack baremetal node boot device set.
- only provide one OpenStackClient command to do something; avoid aliasing
- for naming, the trend is to use Americanised spelling, eg favor instead of favour. Having said that, it is important to take into consideration the terminology/usage outside of OpenStack, e.g. by operators and administrators.

Alternatives

Continue with the current client and remove the existing OSC plugin bits.

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

openstack baremetal chassis

- openstack baremetal chassis show <uuid>

  --fields <field,[field,]> Select fields to fetch and display

- openstack baremetal chassis list

  --long Show detailed chassis info (Mutually exclusive to fields)

  --limit <limit> Limit the number of items to return

  --marker <uuid> Which chassis uuid to start after

  --sort <key[:direction]> Key and direction of sort. <direction> is optional. Defaults to ascending order.

  --fields <field,[field,]> Select fields to fetch and display. (Mutually exclusive to long)

- openstack baremetal chassis create

  --description <description> Chassis description

  --extra <key=value> Extra chassis properties. Can be specified multiple times.

  --uuid <uuid> UUID of the chassis

- openstack baremetal chassis delete <uuid> [<uuid>]
- openstack baremetal chassis set <uuid>

--extra <key=value> Property to set or update. Can be specified multiple times.

--description <description> Chassis description

- openstack baremetal chassis unset <uuid>

  --extra <key> Key of property to unset. Can be specified multiple times.

  --description <description> Will unset the chassis description ()

ironic CLI users who want to see a list of nodes belonging to a given chassis should use `openstack baremetal node list --chassis`, since we will not provide an `openstack baremetal chassis xxx` equivalent to `ironic chassis-node-list`.

openstack baremetal driver

- openstack baremetal driver list
- openstack baremetal driver show <driver>
- openstack baremetal driver passthru list <driver>
- openstack baremetal driver passthru call <driver> <method>

  <method> Vendor passthru method to call.

  --arg <key=value> key=value to add to passthru method. Can be specified multiple times.

  --http-method <http_method> one of POST, PUT, GET, DELETE, PATCH

openstack baremetal node

- openstack baremetal node show <uuid>

  Obsoletes: openstack baremetal show

  --instance Interpret <uuid> as an instance UUID

  --fields <field,[field,]> Select fields to fetch and display.

- openstack baremetal node list

  Obsoletes: openstack baremetal list

  --limit <limit> Limit the number of items to return

  --marker <uuid> Which node to start after

  --sort <key[:direction]> Key and direction of sort. <direction> is optional.

  --maintenance List nodes in maintenance mode

  --associated List nodes associated with an instance

  --chassis <uuid> UUID of chassis to limit node list

  --provision-state <state> Show nodes in specified <state>

  --fields <field,[field,]> Select fields to fetch and display. (Mutually exclusive to long)

- openstack baremetal node create

  Obsoletes: openstack baremetal create

  --chassis-uuid <uuid> Chassis this node belongs to

  --driver <driver> Driver used to control the node

  --driver-info <key=value> key=value pair used by the driver. Can be specified multiple times.

  --property <key=value> Property of the node. Can be specified multiple times.

  --extra <key=value> Arbitrary metadata. Can be specified multiple times.

  --uuid <uuid> Unique UUID of the node. Optional.

  --name <name> Unique name of the node.

- openstack baremetal node delete <uuid> [<uuid>]

  Obsoletes: openstack baremetal delete

- openstack baremetal node set <uuid>

  Obsoletes: openstack baremetal set

  --name <name> Name of the node

  --instance-uuid <uuid> Instance UUID

  --driver <driverid> Driver name or UUID

  --property <key=value> Property to set/update on the node. Can be specified multiple times.

  --extra <key=value> Extra to set/update on the node. Can be specified multiple times.

  --driver-info <key=value> driver-info to set/update on the node. Can be specified multiple times.

  --instance-info <key=value> instance-info to set/update on the node. Can be specified multiple times.

  --target-raid-config <config> Set the target RAID configuration (JSON) for the node. This can be one of: 1. a file containing JSON data of the RAID configuration; 2. - to read the contents from standard input; or 3. a valid JSON string.

- openstack baremetal node unset <uuid>

  Obsoletes: openstack baremetal unset

  --property <key> key to unset on the node. Can be specified multiple times.

  --extra <key> key from extra to unset. Can be specified multiple times.

  --driver-info <key> key to unset from driver-info. Can be specified multiple times.

  --instance-info <key> key to unset from instance-info. Can be specified multiple times.

  --instance-uuid <uuid> Instance uuid.

--name Name of the node.

--target-raid_config target RAID configuration

- openstack baremetal node passthru list <uuid>
- openstack baremetal node passthru call <uuid> <method>

  <method> Vendor passthru method to be called

  --arg <key=value> argument to send to passthru method. Can be specified multiple times.

  --http-method <http_method> One of POST, PUT, GET, DELETE, PATCH
- openstack baremetal node console show <uuid>
- openstack baremetal node console enable <uuid>
- openstack baremetal node console disable <uuid>
- openstack baremetal node boot device show <uuid>

  --supported Show the supported boot devices
- openstack baremetal node boot device set <uuid> <device>

  <device> One of pxe, disk, cdrom, bios, safe

  --persistent Make changes persistent for all future boots.
- openstack baremetal node deploy <uuid>

  --config-drive <config_drive> A gzipped, base64-encoded configuration drive string OR the path to the configuration drive file OR the path to a directory containing the config drive files. In case its a directory, a config drive will be generated from it.
- openstack baremetal node undeploy <uuid>
- openstack baremetal node rebuild <uuid>
- openstack baremetal node inspect <uuid>
- openstack baremetal node provide <uuid>
- openstack baremetal node manage <uuid>
- openstack baremetal node abort <uuid>
- openstack baremetal node maintenance set <uuid>

  --reason <reason> Reason for setting to maintenance mode
- openstack baremetal node maintenance unset <uuid>
- openstack baremetal node power on <uuid>
- openstack baremetal node power off <uuid>
- openstack baremetal node reboot <uuid>
- openstack baremetal node validate <uuid>

ironic CLI users who want to see a list of ports belonging to a given node should use `openstack baremetal port list --node`, since we will not provide an `openstack baremetal node xxx` equivalent to `ironic node-port-list`.

ironic CLI users who want the equivalent to `ironic node-show-states` should use the following command:

```
openstack baremetal node show <node> --fields console_enabled last_error
power_state provision_state provision_updated_at raid_config
target_power_state target_provision_state target_raid_config
```

openstack baremetal port

- openstack baremetal port show <uuid|mac>

  --address <mac> Mac address instead of uuid

  --fields <field[,field,]> Fields to display

- openstack baremetal port list

  --limit <limit> Limit the number of items to return

  --marker <marker> Which port to start after

  --sort <key[:direction]> Key and direction of sort

  --long Display detailed information about ports. Mutually exclusive with fields.

  --fields <field[,field,]> Fields to display. Mutually exclusive with long.

  --node <nodeid> UUID or name of node to limit the port display

- openstack baremetal port create <address>

  --node <uuid> Node uuid to add the port to

  --extra <key=value> Arbitrary key=value metadata. Can be specified multiple times.

- openstack baremetal port delete <uuid> [<uuid>]

- openstack baremetal port set <uuid>

  --extra <key=value> property to set. Can be specified multiple times.

  --address <macaddress> Set new MAC address of port

  --node <nodeid> Set UUID or name of node the port is assigned to

- openstack baremetal port unset <uuid>

  --extra <key> key to remove. Can be specified multiple times.

Not addressed

OpenStackClient commands corresponding to these ironic CLI commands are not addressed by this proposal. They will be addressed in a future release.

- **`ironic driver-raid-logical-disk-properties`. Get RAID logical disk** properties for a driver.

- **`ironic driver-properties`. Get properties (node.driver_info keys and** descriptions) for a driver.

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

N/A

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

None

Developer impact

None

Implementation

Assignee(s)

Primary assignee:

- brad-9 <brad@redhat.com>

Other contributors:

- Romanenko_K <kromanenko@mirantis.com>

- rloo <ruby.loo@intel.com>

Work Items

TBD

Dependencies

None

Testing

Unittests will be added.

Upgrades and Backwards Compatibility

There is already an implementation of some of these commands. A few are likely to change with this spec. These existing commands will go through a deprecation period.

Documentation Impact

The command line documentation will be updated to show these new commands.

References

Third Party Driver Continuous Integration Testing

https://bugs.launchpad.net/ironic/+bug/1526410

This spec is to define the deadlines, requirements and process of bringing third party continuous integration testing to ironic.

Problem description

The ironic project wants to make sure that all the drivers in tree are maintained and of high quality. In order to do this, it has become necessary to require that all drivers have a continuous integration test system to test the driver against every code change proposed for ironic, unless that change is to code or documentation that can not impact the driver. Required tests to be run by each driver test system will be documented in the ironic CI requirements documentation. Not all drivers are provided or maintained by third party vendors.

Third party vendors must provide and maintain this CI for drivers they maintain. Any driver whose maintainer is not able to implement a reliable CI system to test will be removed from the ironic tree.

A driver test system will be deemed reliable if it runs the expected tests and reports the results of those tests consistently over a period of time. Tests will be expected to complete and report back to gerrit within 8 hours of the patch submission until the end of Newton release and within 4 hours by the end of Ocata development cycle.

A driver test system will be deemed reliable if:

- It runs the expected tests for every patch set that is not excluded. Reasons for this exclusion will be documented and approved by the ironic team.
- It reports the results within the expected time frame, see above.

- All required test artfacts are posted in a standard format and made available to the community following the infrastructure requirements.[3]
- Test results (pass or fail) are accurate.

Proposed change

The ironic team has decided to require all driver maintainers to run and maintain CI test systems in order to have their driver code remain in the ironic source tree. When there are problems with the CI system, the driver test team will make a best effort to respond to questions about their system and to fix the system quickly. If the driver test team is not responsive, any voting privileges for the test system may be removed, and may eventually result in the driver being removed from the source tree.

Timeline

The process to implement and start enforcing third party driver CI is to be completed by the end of the Newton development cycle.

Deliverable milestones:

- Immediately: Driver teams contacted and notified of expectations.
- Mitaka-2 milestone: Driver teams will have registered their intent to run CI by creating system accounts and identifying a point of contact for their CI team.
- Mitaka Feature Freeze: All driver systems show the ability to receive events and post comments in the third party CI sandbox.
- N Feature Freeze: Per patch testing and posting comments.

Please refer to the Mitaka release schedule[5] for specific dates. Note that the ironic project does not strictly follow the official OpenStack release cycle deadlines, but we are using the official deadlines as a reference point.

Infra CI will continue to test the ssh drivers already being used in the gate.

Third party driver teams that do not implement a reliable reporting CI test system by the Newton release feature freeze (see Deliverable milestones above) will be removed from the ironic source tree. Driver test systems that miss any of the milestones may be subject to immediate removal from the source tree.

Driver test systems will be required to initially run a test similar to the *gate-tempest-dsvm-ironic-pxe_ipa* test, with the only difference being changes to drivers loaded, etc, to make ironic work with the driver under test.

More information about this and other required tests will need further documentation.

Initial driver test systems will not be allowed to vote on changes to ironic. A driver test team may ask that their CI system be allowed to vote only if the ironic team approves based on the test teams participation, availability and history of reliable operation.

Driver test systems will be expected to follow the Infrastructure Third Party test requirements[3], unless overridden here or in the ironic third party driver testing documentation produced as a result of this spec.

Driver test systems must test patches to master, and patches to branches that were created since the third party CI began testing. For example, if a CI began testing during the Mitaka cycle, then during the N cycle that CI will be expected to test changes to the master and stable/mitaka branches, but not the stable/liberty branch.

Community-maintained drivers that do not have CI testing will also be removed from the ironic source tree.

From Newton feature freeze forward, all new ironic drivers must show that they have a system performing CI testing and meet all infra [3] and ironic requirements in order to land code into the ironic tree. Drivers in progress before Newton feature freeze have until Newton feature freeze to meet these requirements. CI testing does not need to be shown in order to propose a driver spec, but must be in place before landing code.

Alternatives

None

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

N/A

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

When upgrading to the release that drops untested drivers, if a deployer is using a driver that is removed from the tree, they will need to change to an in-tree driver or install the removed driver from a new location, if one exists.

The Ironic team must communicate which drivers are being removed, and when. We should note that these drivers *may* be available at a new location, and that driver authors *may* be communicating that information.

Authors of a driver removed from tree may communicate the new location, if one exists, and document how to install the driver into an ironic environment.

Developer impact

Developer impacts may include core reviewers needing to wait until testing for a system completes before approving a patch for merge. Developers that had a test fail will need to review the test artifacts for their patch linked to the comment left in the patch comment log. If necessary, the developer may need to coordinate with the driver test team for help with debugging the problem. See Infra requirements in the References section below.

Implementation

Assignee(s)

Primary assignee:
krtaylor

Other contributors:
jroll, thingee

Work Items

1. Communicate intention to vendors with existing drivers in tree - make a reasonable effort to contact the entity responsible for the driver and inform them of the timeline to require driver third party CI.
2. Set incremental timeline milestones for vendors to implement CI testing.
3. A deprecation process will need to be documented.
4. Document process, requirements - this spec is not meant to exhaustively enumerate all requirements, just to define that they need to be documented.

5. The documentation will also need to describe the way in which a test system proves they are adequately testing their driver.
6. Assemble and maintain list of contacts for all in-tree drivers.
7. Remove third party drivers that do not implement a CI test system as per the schedule listed above.
8. Document impacts to ironic deployers and developers that the driver they may have been using was removed from tree, as per the deployer impact section.

Dependencies

None

Testing

As described in this spec.

Upgrades and Backwards Compatibility

There will be a major upgrade impact on deployers using drivers that are removed from tree; see the Deployer impact section for more info.

A deprecation process will be documented including timeline.

Documentation Impact

There will be several areas impacted:

1. Document drivers in tree and their expected functionality.
2. Document requirements for the third party drivers systems, expectations, time thresholds, tests required to be run, and other topics as needed.
3. Document an example implementation of the third party test system infrastructure.
4. Document the process to notify the community and users that a driver will be removed from tree.
5. Document more information about the required tests

References

[1] Third Party CI working group https://wiki.openstack.org/wiki/ThirdPartyCIWorkingGroup

[2] Third party CI meetings https://wiki.openstack.org/wiki/Meetings/ThirdParty

[3] Infra requirements documentation for implementing a third party system http://docs.openstack.org/infra/system-config/third_party.html#requirements

[4] Discussion at Mitaka summit https://etherpad.openstack.org/p/summit-mitaka-ironic-third-party-ci

[5] Mitaka release schedule: https://wiki.openstack.org/wiki/Mitaka_Release_Schedule

5.11.2 6.1

Add pluggable metrics backend for Ironic and IPA

https://bugs.launchpad.net/ironic/+bug/1526219

This proposes the addition of metric data reporting features to Ironic, and Ironic Python Agent (IPA). Initially, this will include a statsd reference implementation, but will be sufficiently generic to permit the creation of alternative backends.

Problem description

Software metrics are extremely useful to operators for recognizing and diagnosing problems in running software, and can be used to monitor the real time and historical performance of Ironic and IPA in a production environment.

Metrics can be used to determine how quickly (or slowly) parts of the system are running, how often errors (such as API error responses or BMC failures) occur, or the performance impact of a given change.

Currently, neither Ironic nor IPA report any application metrics.

Proposed change

- Design a shared pluggable metric reporting system.
- Implement a generic MetricsLogger which includes:
  - Gauges (generic numerical data).
  - Counters (increment/decrement a counter).
  - Timers (time something).
  - Decorators, and context managers for same.
- Implement a StatsdMetricsLogger as the reference backend [1].
- Instrument Ironic to report metric data including:
  - Counting and timing of API requests. This may be accomplished by hooking into Pecan.
  - Counting and timing of RPCs.
  - Counting and timing of most worker functions in ConductorManager.
  - Counting and timing of important driver functions.
  - Count and time node state changes. By inspecting provision_updated_at during a state change, the time the node spent in that state can be calculated.
- Instrument IPA to report metric data including, but not limited to:
  - Image download/write counts and times.
  - Deploy/cleaning counts and times.

Example code follows (based on Python logging module naming conventions):

```
METRICS = metrics.getLogger(__name__)

class Foo(object):
  def func1(self):
    # Emit gauge metric with value 1
    METRICS.send_gauge("one.fish", 1)

    # Increment counter metric by two
```

(continues on next page)

(continued from previous page)

```
    METRICS.send_counter("two.fish", 2)

    # Decrement counter metric by one
    METRICS.send_counter("red.fish", -1)

    # Randomly sample the data (emit metric 10% of the time)
    METRICS.send_counter("blue.fish", 42, sample_rate=0.1)

    # Emit a timer metric with value of 125 (milliseconds)
    METRICS.send_timer("black.fish", 125)

    # Randomly sample the data (emit metric 1% of the time)
    METRICS.send_timer("blue.fish", 125, sample_rate=0.01)

@METRICS.counter("func2.count")
@METRICS.timer("func2.time", sample_rate=0.1)
def func2(self):
  pass

# Context managers for counting and timing code blocks
def func3(self):

  with METRICS.counter("func3.thing_one.count", sample_rate=0.25):
    thing_one()

  with METRICS.timer("func3.thing_two.time"):
    thing_two()
```

Metric names follow this convention (optional parts indicated by []):

`[global_prefix.][host_name.]prefix.metric_name`

If *metrics-agent-prepend-host-reverse* is set, then `host.example.com` becomes `com.example.host` to assist with hierarchical data representation.

For example, using the Statsd backend, and relevant config options, `METRICS.send_timer("blue.fish", 125, sample_rate=0.25)` is emitted to statsd as `globalprefix.com.example.host.moduleprefix.blue.fish:1|ms@0.25`.

Alternatives

Alternatively, we could implement a Ceilometer backend. Although Ironic already reports some measurements (such as IPMI sensor data) to Ceilometer, the metrics that are proposed in this spec do not fit with the Ceilometer project mission, which is to collect measurements of the utilization of the physical and virtual resources comprising deployed clouds [2]

Instead, this spec proposes that we instrument parts of the Ironic/IPA codebase itself to report metrics and statistics about how/when the code is run, and the performance of the code thereof. This data is not directly related to physical and virtual resources comprising deployed clouds. Therefore, we do not propose the addition of a Ceilometer backend, nor do we propose that the existing Ceilometer measurements be converted to this system, as they represent fundamentally different types of data.

Data model impact

None

State Machine Impact

None.

REST API impact

To support agent drivers, a config field will be added to the response for the `/drivers/<drivername>/vendor_passthru/lookup` endpoint in the Ironic API.

This field will contain the agent-related config options that an agent can use to configure itself to report metric data. For example: statsd host and statsd port.

Client (CLI) impact

None.

RPC API impact

None.

Driver API impact

None.

Nova driver impact

None.

Ramdisk impact

N/A

Security impact

The statsd daemon [3] has no authentication, and consequently anyone who is able to send UDP datagrams to the daemon can send arbitrary metric data. However, the statsd daemon is typically configured to listen only on a local interface, which partially mitigates security concerns.

Other end user impact

None.

Scalability impact

Deployers must ensure that their statsd infrastructure is scaled correctly relative to the size of their deployment. However, even if the statsd daemon is overloaded, Ironic will not be negatively affected (statsd UDP datagrams are non-blocking, and will simply not be processed).

Performance Impact

By default, metrics reporting will be disabled, reducing, but not totally eliminating, the performance impact for users who do not wish to collect metrics. At the very least, a conditional must be checked at each place where a metric could be reported. Furthermore, depending on exactly how and where the conditional checking occurs, arguments may be evaluated even if the metric data arent actually sent.

Reporting metrics via statsd affects performance minimally. The overhead of sending a single piece of metric data is very smallin particular, statsd metrics are sent via UDP (non-blocking) to a daemon [2] that aggregates the metrics before forwarding them to one of its supported backends. Should this backend become unresponsive or overloaded, then metric data will be lost, but without other performance effects.

After the metric data are aggregated by a local statsd daemon, they are periodically flushed to one of statsds configured backends, usually Graphite [4].

Other deployer impact

There are two different sets of configuration options to be added:

These options will be set in the ironic-lib metrics library, and will be used by any ironic service implementing metrics:

```
[metrics]

# Backend options are "statsd" and "noop"
backend="noop"
statsd_host="localhost"
statsd_port=8125

# See proposed changes section for detailed description of how these are used
prepend_host=false
prepend_host_reverse=false
global_prefix=""
```

Additionally, the following options will be added to the ironic-conductor and used to configure the ironic-python-agent for metrics on lookup:

```
# Backend options are "statsd" and "noop"
agent_backend="noop"
agent_statsd_host="localhost"
agent_statsd_port=8125

# See proposed changes section for detailed description of how these are used
agent_prepend_host=false
agent_prepend_host_reverse=false
agent_prepend_uuid=false
agent_global_prefix=""
```

If the statsd metrics backend is enabled, then deployers must install and configure statsd, as well as any other metrics software that they wish to use (such as Graphite [3]). Additionally, if deployers wish to emit metrics from ironic-python-agent as well, the statsd backend must be accessible from networks that agents run on.

Developer impact

None.

Implementation

Assignee(s)

Primary assignee:
alineb

Other contributors:
JayF

Work Items

- Design/implement metric reporting into ironic-lib.
- Implement statsd backend.
- Instrument Ironic code to report metrics.
- Instrument IPA code to report metrics.

Dependencies

None.

Testing

Additional care may be required to test the statsd network code.

Upgrades and Backwards Compatibility

None.

Documentation Impact

Appropriate documentation must be written.

References

For more on why metrics are useful to operators, and why the statsd project began: https://codeascraft.com/2011/02/15/measure-anything-measure-everything/

[1] https://github.com/etsy/statsd/blob/master/docs/metric_types.md

[2] https://wiki.openstack.org/wiki/Ceilometer

[3] https://github.com/etsy/statsd/

[4] https://graphite.readthedocs.org/en/latest/faq.html

Promote agent vendor passthru to core API

https://bugs.launchpad.net/ironic/+bug/1570841

This spec suggests making the current agent vendor passthru API (lookup and heartbeat) first class API endpoints and deprecate the agent vendor passthru interface.

Problem description

The vendor passthru was designed for vendors to place their specific features before they get wider adoption in Ironic and get promoted to the core API.

However, when IPA is used (which is the only ramdisk available in-tree), these two API endpoints play the critical role in both deployment and cleaning processes. Thus every IPA-based driver must mix agent vendor passthru into its vendor passthru.

There was a bug in the drac driver when it was not done, and the driver did not work with IPA.

This proposal also tries to reduce the amount of data sent to and from unauthenticated endpoints. The current vendor passthru API accepts the whole inventory and returns the whole node record, including IPMI credentials.

Proposed change

- Create new API endpoints for lookup and heartbeat - see *REST API impact* for details.
- Extend the deploy interface with a heartbeat method - see *Driver API impact* for details.

Alternatives

- Continue doing what we do now.
- Make the lookup call driver-dependent (as the passthru used to be).

  This looks like unnecessary complication (e.g. we have to pass a driver to IPA from the conductor nowadays).
- We could use this change to move the unauthenticated endpoints away from the main API completely. This could be done by introducing a new API service, say `ironic-agent-api`, serving only these two endpoints. Then we will recommend operators to make this service only listen on the provisioning network, but not on the network accessible to users.

  Arguably the same result can be achieved by configuring a WSGI container (Apache mod_wsgi or similar), so it might not be worth complication.

Data model impact

None

State Machine Impact

None

REST API impact

Two new endpoints are added. Both endpoints are **NOT** authenticated.

- `GET /v1/lookup?addresses=MAC1,MAC2&node_uuid=UUID`

  Look up node details for further use in the ramdisk.

  No body; at least one of the following URL parameters must be present:

  - `addresses` comma-separated list of hardware addresses (e.g. MAC) from the node for lookup;
  - `node_uuid` node UUID, if known (e.g. by inspection).

  If `node_uuid` is present, `addresses` are ignored.

  By default only return a node if its in one of transient states: `deploying`, `deploy wait`, `cleaning`, `clean wait`, `inspecting`, `inspect wait`. Deployers who need lookup to always work will be able to set a new option `[api]restrict_lookup` to `False`.

  Note

  In theory, we dont need `-ing` states here either. But when we reboot during cleaning, we dont currently reset the state to `clean wait`. The other `-ing` states are supported in case 3rd party drivers have similar restrictions.

  Response: HTTP 200 with JSON body containing keys:

  - `config` dictionary for passing configuration options from conductor to the ramdisk. For the IPA ramdisk only one is currently used:
    - `heartbeat_timeout` timeout (in seconds) between heart beats from the ramdisk, expected by Ironic.
  - `node` partial node representation as a JSON object, with the following fields sent:
    - `properties` for root device hints,
    - `instance_info` for disk sizing details,
    - `uuid` node UUID,
    - `driver_internal_info` for passing other runtime information.

    More fields can be exposed with time with an appropriate API version bump.

  Error codes:

  - 400 - bad request,
  - 404 - a node was not found.

- `POST /v1/heartbeat/<UUID>`

  Record a heartbeat message from the ramdisk.

  Body is a JSON with fields:

  - `callback_url` - the IPA URL to call back. Note that for potential non-IPA-based drivers it might have a different meaning (e.g. if we agree on the ansible driver, this can be an SSH URL for it).

Response: HTTP 202 with no body.

Error codes:

- 400 - bad request,
- 404 - a node was not found,
- 409 - node is locked (should be retried by the ramdisk).

A new API version will be introduced to cover both endpoints.

Client (CLI) impact

Both endpoints will be exposed in the Python API for the ironic client as:

```
ironic.node.lookup(addresses, node_uuid=None)
ironic.node.heartbeat(node_uuid, callback_url)
```

However, as they are not intended for end users, they will not be exposed in both CLI.

ironic CLI

None

openstack baremetal CLI

None

RPC API impact

A new RPC call is created to connect the heart beat API endpoint and the new deploy driver method: `heartbeat` (async).

Driver API impact

A new method is added to the deploy driver interface:

```
def heartbeat(self, task, callback_url):
    """Record a heart beat for the node.

    :param task: a task manager task
    :param callback_url: a URL to use to call to the ramdisk
    :return: None
    """
    LOG.warning('Got heartbeat message from node %(node)s, but the driver '
                '%(driver)s does not support heartbeating',
                {'node': task.node.uuid, 'driver': task.node.driver})
```

The heartbeat method from `BaseAgentVendor` will be refactored to a separate mix-in class for reusing in both `AgentDeploy` and `BaseAgentVendor`.

The new method will not be abstract to allow drivers that use a different approach (e.g. which do not have a ramdisk at all). The default implementation will do nothing to account for deploy drivers which do not need heart beats.

The new method will receive a shared node lock. It is up to the implementation to upgrade the lock to exclusive, if required.

Nova driver impact

None

Ramdisk impact

None

Security impact

- This change will expose unauthenticated API to lookup a node by its MAC addresses. It does not have any impact on most deployments, as both in-tree deploy methods (iscsi and http) already expose such API.
- After the complete switch to the new API endpoints is finished, it will no longer be possible to fetch the whole node knowing its MAC address without authentication. Only limited fields will be available. Notably, the power credentials are not sent in the new API endpoints.
- We should clearly note that any deploy implementation should treat the incoming data in the new `heartbeat` call with care. Particularly, no sensitive information should be ever sent to the endpoint designated by the `callback_url` parameter.

Other end user impact

None

Scalability impact

None

Performance Impact

- Unlike the old lookup passthru, the new lookup endpoint will not use RPC, lowering load on the message queue and the conductor.

Other deployer impact

- An update of the IPA image will be recommended to make it use the new API.
- New option `restrict_lookup` in the `api` section (boolean, defaults to `True`) - whether to restrict the new lookup API to only certain states in which lookup is expected.

Developer impact

3rd party driver developers should stop using the `BaseAgentVendor` class in their drivers and just use the `AgentDeploy` class.

3rd party drivers should document whether they require the `restrict_lookup` option to be `False` for correct functioning.

Implementation

Assignee(s)

- Dmitry Tantsur (lp: divius, irc: dtanstur)
- Jim Rollenhagen (irc: jroll)

Work Items

- Create new deploy interface methods
- Implement them in the AgentDeploy
- Create new RPC calls and API endpoints
- Switch IPA to use the new endpoints, and fall back to old ones on failure

Dependencies

None

Testing

Testing will be conducted as part of the current gate tests.

Upgrades and Backwards Compatibility

Full backward compatibility will be guaranteed independent of upgrade order between IPA and ironic itself.

The `BaseAgentVendor` class will be deprecated, but stay for some time, following the usual deprecation policy. Old IPA images will be able to run by using the old passthru API.

The new IPA image will try to hit the new endpoints first, and will fall back to the old ones on getting HTTP 406 Not Acceptable (meaning, the API version is not supported).

Documentation Impact

- Document how to implement new deploy drivers with the new heartbeat method.
- Document the potential security issues with both endpoints.

References

Collect system logs from IPA

https://bugs.launchpad.net/ironic/+bug/1587143

This spec adds support for retrieving the deployment system logs from IPA.

Problem description

We currently have no mechanism to automatically retrieve the system logs from the IPA deploy ramdisk. Having access to the logs may be very useful, especially when troubleshooting a deployment failure. Currently, there are a few ways to get access to the logs in the ramdisk, but they are manual, and sometimes it is not desirable to enable them in production. The following points describe two of them:

1. Have a console session enabled for the node being deployed.

   While this works, its tricky because the operator needs to figure out which node was picked by the scheduler and enable the console for it. Also, not all drivers have console support.

2. Disable powering off a node upon a deployment failure.

   Operators could disable powering off a node upon a deployment failure but this has some implications:

   a. It does not work in conjunction with Nova. When the instance fails to be provisioned nova will invoke destroy() and the Ironic virt driver will then force a power off on that node.

   b. Leaving the nodes powered on after the failure is not desirable in some deployments.

Proposed change

The proposed implementation consists in having Ironic retrieve the system logs from the deploy ramdisk (IPA) via its API and then upload it to Swift or save it on the local file-system of that conductor (for standalone-mode users).

Changes in IPA

A new `log` extension will be added to IPA. This extension will introduce a new synchronous command called `collect_system_logs`. By invoking this command IPA will then tar, gzip and base64 encode the system logs and return the resulting string to the caller.

Since we do support different base OSs for IPA (e.g Tiny Core Linux, Fedora, Debian) we need different ways to find the logs depending on the system. This spec proposes two ways that should be enough for most of the distros today:

1. For distributions using `systemd`, all system logs are available via `journald`. IPA will then invoke the `journalctl` command and get the logs from there.
2. For other distributions, this spec proposes retaining all the logs from */var/log* and the output of the `dmesg` command.

The logs from all distributions independent of the init system, will also contain the output of the following commands files: `ps`, `df`, and `iptables`.

Changes in Ironic

New configuration options will be added to Ironic under the `[agent]` group:

1. `deploy_logs_collect` (string): Whether Ironic should collect the deployment logs or not. Valid options are: always, on_failure or never. Defaults to on_failure.
2. `deploy_logs_storage_backend` (string): The name of the storage backend where the response file will be stored. One of the two: local or swift. Defaults to local.
3. `deploy_logs_local_path` (string): The path to the directory where the logs should be stored, used when the `deploy_logs_storage_backend` is configured to **local**. Defaults to `/var/log/ironic/deploy`.
4. `deploy_logs_swift_container` (string): The name of the Swift container to store the logs, used when the `deploy_logs_storage_backend` is configured to **swift**. Defaults to *ironic_deploy_logs_container*.

5. `deploy_logs_swift_days_to_expire` (integer): Number of days before a log object is marked as expired in Swift. If None, the logs will be kept forever or until manually deleted. Used when the `deploy_logs_storage_backend` is configured to **swift**. Defaults to *30 days*.

Note

When storing the logs in the local file-system Ironic wont be responsible for deleting the logs after a certain time. Its up to the operator to configure an external job to do it, if wanted.

Depending on the value of the `deploy_logs_collect` Ironic will invoke `log.collect_system_logs` as part of the deployment of the node (right before powering it off or rebooting). For example, if `deploy_logs_collect` is set to **always** Ironic will collect the logs independently of the deployment being a success or a failure; if it is set to **on_failure** Ironic will collect the logs upon a deployment failure; if it is set to **never**, Ironic never collect the deployment logs.

When the logs are collected, Ironic should decode the base64 encoded tar.gz file and store it according to the `deploy_logs_storage_backend` configuration. All log objects will be named with the following pattern: *<node-uuid>[_<instance-uuid>]_<timestamp yyyy-mm-dd-hh:mm:ss>.tar.gz*. Note that, `instance_uuid` is not a required field for deploying a node when Ironic is configured to be used in **standalone** mode so, if present it will be appended to the name.

When using Swift, operators can associate the objects in the container with the nodes in Ironic and search for the logs of a specific node using the `prefix` parameter, for example:

```
$ swift list ironic_deploy_logs_container -p 5e9258c4-cfda-40b6-86e2-
↪e192f523d668
5e9258c4-cfda-40b6-86e2-e192f523d668_0c1e1a65-6af0-4cb7-a16e-8f9a45144b47_
↪2016-05-31_22:05:59
5e9258c4-cfda-40b6-86e2-e192f523d668_db87f2c5-7a9a-48c2-9a76-604287257c1b_
↪2016-05-31_22:07:25
```

Note

This implementation requires the network to be setup correctly, otherwise Ironic will not be able to contact the IPA API. When debugging such problems, the only action possible is to look at the consoles of the nodes to see some logs. This method has some caveats: see the *Problem description* for more information.

Note

Neither Ironic or IPA will be responsible for sanitizing any logs before storing them. First because this spec is limited to collecting logs from the deployment only and at this point the tenant wont have used the node yet. Second, the services generating the logs should be responsible for masking secrets in their logs (like we do in Ironic), if not, it should be considered a bug.

Alternatives

Since we already provide ways of doing that via accessing the console or disabling the powering off the nodes on failures, there are few alternatives left for this work.

The current proposed solution could be extended to fit more use cases beyond what this spec proposes. For example, instead of uploading it to Swift or storing it in the local file-system, Ironic could upload it to a HTTP/FTP server.

As briefly described at *Changes in IPA* the method to collect the logs could be extended to include more logs and output of different commands that are useful for troubleshooting.

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

None

Security impact

None.

As a note, credentials **are not** passed from Ironic to the deploy ramdisk. The `ironic-conductor` service, which already holds the Swift credentials, is the one responsible for uploading the logs to Swift.

Other end user impact

None

Scalability impact

None

Performance Impact

The node will stay a little longer in the `deploying` provision state while IPA is collecting the logs, if enabled.

Other deployer impact

None

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
lucasagomes <lucasagomes@gmail.com>

Other contributors:

Work Items

- Add the new `log` extension and `collect_system_logs` method in IPA.
- Add the new configuration options described in the *Changes in Ironic* section.
- Invoke the new `log.collect_system_logs` method in IPA as part of the deployment and store the response file according to the `deploy_logs_storage_backend` configuration option (if enabled).

Dependencies

None

Testing

Unittests will be added.

Upgrades and Backwards Compatibility

None.

As a note, when using an old IPA ramdisk which does not support the new `log.collect_system_logs` command Ironic should handle such exception and log a warning message to the operator if `deploy_logs_collect` is set to **always** or **on_failure**.

Documentation Impact

Documentation will be provided about how to configure Ironic to collect the system logs from the deploy ramdisk.

References

None.

Keystone Policy Support

https://bugs.launchpad.net/ironic/+bug/1526752

Keystone and oslo.policy have support for restricting access based on information about the user authenticating, allowing partial access to be granted as configured by operators. This spec lays out how this support will be implemented in ironic.

Problem description

Ironic has traditionally operated on an all-or-nothing access system, only restricting access to passwords. This model is significantly limited when multiple people and groups with different trust levels want to interact with ironic. For example, a hardware technician may need access to set or unset maintenance on the node, but should not have access to provision nodes.

Proposed change

- Ensure proper metadata, such as role, is derived from the auth_token when authenticating by properly implementing KeystoneMiddleware.auth_token support.
- Define policy rules for each RESTful action on each API endpoint, scoped to the baremetal namespace.
- Configure each API endpoint to verify a user is permitted by policy to access it.
- Implement specific restrictions for sensitive information, including configdrives and passwords. Default to hide all sensitive information.
- Define sane default policies in code[0] , with shipped roles including an admin role with full access and an observer role with read-only access to non-secret information. Names for these roles will be determined during implementation. A sample policy.json[1] shall be generatable using oslopolicy-sample-generator.
- Maintain compatibility with all roles in the previously-shipped policy.json configuration file.

Alternatives

A deployer could implement ironic behind a reverse proxy and use another authentication method to allow or disallow access based on path and HTTP method. This is onerous, does not follow the pattern set by other OpenStack services, and does not provide the granularity that properly implementing policy support would.

[0] Oslo Policy in Code https://specs.openstack.org/openstack/oslo-specs/specs/newton/policy-in-code.html
[1] Policy JSON syntax http://docs.openstack.org/kilo/config-reference/content/policy-json-file.html

Data model impact

None.

State Machine Impact

Users may be restricted by policy from moving nodes within the state machine. However, there are no direct state machine modifications.

REST API impact

A properly restricted user may receive a 403 error if they are unable to use the method/endpoint combination requested. However, the REST API will not be returning 403 in any case it could not today, for instance, an unauthorized user may receive 403 today. This simply increases the granularity available for configuring this authorization.

The 403 response body shall indicate which resource access was denied to.

Client (CLI) impact

A CLI client user will need to have a properly authorized user to perform any requested actions.

RPC API impact

None.

Driver API impact

Drivers can now enforce policy within any driver_vendor_passthru methods as desired.

Nova driver impact

Existing deployments can continue to use a full-admin user as required prior to this feature. Once upgraded, a deployer could use a less-privileged user for nova-ironic interactions.

Ramdisk impact

N/A

Security impact

This changes primary impact is around improving the security of the system. Deployers of ironic will no longer need to provide an admin credential to manipulate only a small part of ironics API.

Other end user impact

None.

Scalability impact

None.

Performance Impact

Policy support is a minimal increase in overhead. Additionally, most policies will be implemented early in the API layer, to prevent ironic from doing excessive work before a user is deemed unauthorized.

Other deployer impact

Deployers will now be able to configure policies, in the policy.json DSL[Page 582, 1] , to meet their specific needs.

Developer impact

Whenever a developer implements a new API method, they will be required to add a new policy rule to represent that API endpoint or method, define the default rule, enforce the policy appropriately, and update default policy as necessary.

Implementation

Assignee(s)

Primary assignee:
devananda

Other contributors:
JayF

Work Items

- Update devstack to configure users properly.
- Change configuration of nova in devstack to use new baremetal_driver role.
- Document how to utilize policy, including how to create users in keystone and assign them to the baremetal project.
- Document any differences in how this impacts users of Keystone API v2 vs v3.

Dependencies

None.

Testing

- Grenade testing to ensure we do not break existing deployments.
- Unit testing to ensure policies are being properly enforced.

Upgrades and Backwards Compatibility

Existing deployers are required to use an admin user for all uses of ironic, these users will continue to have full access to the ironic API, allowing for backwards compatibility.

On upgrade, an operator must define new keystone roles and assign these to users in order to take advantage of the new policy support. The names for these roles will be determined during implementation.

The operator may choose to customize the policy settings for their deployment.

Documentation Impact

- Default policies will need to be documented.
- Install guide will need to be updated with instructions on how to create users with proper roles and project membership.
- Documentation must be written instructing users how to utilize the new policy functionality on upgrade.

References

Pluggable network providers

https://bugs.launchpad.net/ironic/+bug/1526401

Today, Ironic provisions servers on the same (flat) network that tenants run on. Ironic should have the ability to logically separate provisioning and tenant networks, and switch between them as needed.

Note: where this spec says network, it generally means Neutron network, which may be implemented in different ways.

Problem description

Ironic currently provisions servers on the same (flat) network that tenants use. It is clear that Ironic should allow operators to separate provisioning and tenant networks; where deploy ramdisks run on the provisioning network, and instances run on one or more tenant networks.

In order to secure the provisioning network, Ironic should be able to switch between these networks when spawning and destroying instances.

This method should also be extended to other management networks that may or may not be separated from the provisioning network; for example the operator may wish to use different management networks for cleaning or rescue tasks.

Proposed change

Ironic should have a pluggable network provider that can handle switching nodes between different networks. This provider should be able to be selected per Ironic node object. A new *network_provider* field will be added to the node object to define which network provider to use for that node. There should also be a configuration option for the default network provider, defaulting to none for now. The default value for *node.network_provider* will be NULL, meaning use the configuration option. Both the node field and the configuration option may be set to none or neutron, mapping to the no-op provider and the Neutron provider, respectively.

This network provider system will not be part of Ironics driver interface mixin system; rather it is standalone and is loaded on demand via stevedore. This is similar to how the DHCP provider code works today.

An initial implementation of a no-op network provider should be written, to put the logic in place in Ironic while maintaining compatibility with the existing model.

A second implementation should be written that uses Neutron to attach hardware to networks as needed, and should work as follows.

The network provider should have a concept of a provisioning network, where the control plane can communicate with nodes managed by Ironic in order to deploy, tear down, or otherwise manage nodes.

It should be able to connect and disconnect nodes from this network.

The network provider should also know how to connect and disconnect nodes from arbitrary tenant networks. These networks are defined by the Nova user and Nova. Nova should continue to create Neutron ports (a logical attachment to a network), but these ports will be left unbound, as they will not have enough information to be plumbed through. Ironic later should send a port-update call to Neutron, to pass the necessary information to complete the binding. This call should happen after deploying the image, between the power off and power on calls that boot to the instance image. This may have implications for boot from volume workloads. As Ironic does not yet support these workloads, these are out of scope for the purposes of this spec.

In the case where the Ironic driver is used, Nova should send a null `host_id` in the binding profile. This will prevent Neutron from binding the port immediately, so we can defer this and allow Ironic to tell Neutron to bind the port when it is ready to do so. Ironic should send the node UUID as the `host_id`. Ironic will also delete the Neutron port connecting the node to the provisioning network at this time. The reverse will happen at tear down time.

If an older client (e.g. Nova) is in use and does initially send the `host_id`, Ironic needs to handle this. There are two cases here:

- The node is using the Neutron network provider. In this case, Ironic should fetch the ports first, and if the ports are already bound with the correct info, do nothing. If binding failed due to missing switchport information, Ironic should update the port appropriately and allow it to be bound.
- The node is using the none network provider. In this case, the node is expected to be on the provisioning network after deployment (todays current model). In this case, the ports should be treated as they are today, putting DHCP configs on those ports, etc.

Nova and Ironic should both use the binding:profile dictionary to send data such as physical switchport information.

Nova currently has a broken assumption that each Ironic port (physical NIC) may only attach to one network. This assumption will need to be fixed, as hardware (today) cannot spawn arbitrary NICs like virtual servers can. We may, in the future, also need a way for Nova users to define which NIC is attached to which networks. A first version should leave this assumption in place for the sake of simplicity.

If port groups exist for a node, those should be connected to the networks rather than the individual port. This allows for an aggregated connection such as a LAG to connect to the network.

Note that an Ironic environment may be deployed without Nova. In this case, the user should make the same calls Nova would make.

One caveat here is that nodes will not be able to PXE boot the instance image if they cannot reach the conductor (for tftp). Local boot will need to be used for any node deployed outside of the provisioning network. Any deploys outside of the provisioning network that do not use local boot should error.

Deploy drivers should call to this interface at proper times to switch between networks. For example, a driver completing a deploy should power off the node, switch to the instance networks, and power on the node. This will ensure that the deploy ramdisk never runs on the instance network, and the instance image never runs on the provisioning network.

Alternatives

Alternatively, Ironic could continue to prescribe that operators run Ironic on a single flat network shared between tenants and the control plane. This is clearly not viable for many real-world use cases.

Data model impact

A *network_provider* field will be added to the Node object.

State Machine Impact

None.

REST API impact

Update the REST API for the node object to allow reading and modifying the new network_provider field. This will likely need a version bump.

Client (CLI) impact

Will need to update the CLI to print the new Node.network_provider field, when available.

RPC API impact

None.

Driver API impact

This adds a new interface, *NetworkProvider*. This interface is *not* a part of Ironics driver composition system, to be clear. This interface will define the following methods:

```
def add_provisioning_network(self, task):
"""Add the provisioning network to a node."""

def remove_provisioning_network(self, task):
"""Remove the provisioning network from a node."""

def add_cleaning_network(self, task):
"""Add the cleaning network to a node."""

def remove_cleaning_network(self, task):
"""Remove the cleaning network from a node."""

def configure_tenant_networks(self, task):
"""Configure tenant networks (added by Nova/user) for a node."""

def unconfigure_tenant_networks(self, task):
"""Unconfigure tenant networks (to be removed by Nova/user) for a node."""
```

Nova driver impact

The Nova driver should not be directly impacted here; however, this does depend on changes to the Neutron network driver in Nova as described above.

Ramdisk impact

N/A

Security impact

This potentially improves security by restricting tenant access to the control plane.

Other end user impact

To use this feature, end users will need to:

- Set nodes to use the Neutron provider.
- Use local boot for nodes using the Neutron provider.

Scalability impact

When configured to use the Neutron plugin, this will result in additional API calls to Neutron to manage a node. However, impact on scalability should be negligible.

Performance Impact

None.

Other deployer impact

Two new configuration options will be added:

- `CONF.provisioning_network` specifies the ID of the provisioning network.
- `CONF.default_network_provider` specifies the default network provider to use for nodes with *node.network_provider* set to NULL.

A new database column (Node.network_provider) is also added, and so deploying this change will require a database migration to be ran.

Deployers will need to deploy a version of Nova that supports this feature, if using Nova.

Deployers will need to deploy an ML2 mechanism driver that supports connecting baremetal resources to Neutron networks.

Developer impact

Driver authors should support this feature by calling the methods provided.

Implementation

Assignee(s)

jroll <jim@jimrollenhagen.com>

And hopefully many others! :)

Work Items

- Add the Node.network_provider field and the default_network_provider configuration option..
- Implement the base interface.
- Implement the no-op provider.
- Instrument each deploy driver with calls to this interface.
- Implement the Neutron plugin provider.
- Modify Nova to send the extra flag discussed above, when creating ports for a machine using the Ironic virt driver.

Dependencies

None.

Testing

The no-op provider will be tested in the gate by default.

Neutron will provide an ML2 mechanism that simulates connecting real hardware to real switches. When that mechanism is available, we can test the Neutron provider in the gate.

Upgrades and Backwards Compatibility

Default behavior is the current behavior, so this change should be fully backwards compatible.

Documentation Impact

This feature will be fully documented.

References

Discussions on the topic include:

- https://etherpad.openstack.org/p/YVR-neutron-ironic
- https://etherpad.openstack.org/p/liberty-ironic-network-isolation
- Logs from https://wiki.openstack.org/wiki/Meetings/Ironic-neutron
- The spec for the rest of the API and data model changes, and ML2 integration in general: https://review.opendev.org/#/c/188528

Add node resource_class field

https://bugs.launchpad.net/ironic/+bug/1604916

Nova has a plan for scheduling ironic resources the same way as other nova resources, that involves making each ironic node a resource provider, which has a resource class. That resource class is referenced by the nova flavor, in short saying that the flavor requires exactly one thing from that resource class. When running the resource tracker, nova must be able to associate each ironic node with a resource class, which operators must be able to specify (because operators are the people creating the flavors and need to know how to associate the flavor). To allow for this, we add a new resource_class field to ironics node object.

Problem description

Currently, nova tracks ironic resources with a (host, node) tuple. This causes a number of problems within novas internals, and the nova team is trying to get away from this. At the same time, nova wants to schedule ironic instances the same way as other instances are scheduled. This allows ironic to follow along in the ongoing scheduler changes, which will help reduce (or eventually eliminate) scheduling races, and benefit from other scheduler optimizations coming down the road (like making qualitative decisions more performant).

Proposed change

The current proposal in nova is to make each ironic node a resource provider, which is associated with a resource class.[0] A nova flavor may declare that it requires some amount of a given resource class; in ironics case it would require one and only one. The resource provider record for an ironic node will declare that it provides exactly 1 (or 0, if it is in maintenance or similar) of the resource class for that provider record.

This proposal still allows nodes to be scheduled to based on qualitative properties, such as capabilities or affinity rules. The resource class is simply the first filter in this case (though it isnt a traditional scheduler filter).

To do so, nova needs to know which resource class the resource provider record needs to be associated with. As operators manage the flavors that will be linked to these classes, we need a way for the operator to specify what class each node is in, so that the flavor is linked correctly. As such, we add a resource_class field to the node record in ironic, which nova will use when creating the resource provider record.

As an example, imagine an ironic deployment has the following nodes:

```
- node-1:
  resource_class: ironic-gold
  properties:
    cpus: 4
    memory_mb: 16384
    capabilities:
      boot_mode: uefi,bios
- node-2:
  resource_class: ironic-silver
  properties:
    cpus: 2
    memory_mb: 8192
```

The operator might define the flavors as such:

```
- baremetal-gold
  resources:
    ironic-gold: 1
  extra_specs:
    capabilities: boot_mode:bios
- baremetal-gold-uefi
  resources:
    ironic-gold: 1
  extra_specs:
    capabilities: boot_mode:uefi
```

(continues on next page)

(continued from previous page)

```
- baremetal-silver
  resources:
    ironic-silver: 1
```

Note that the flavor definition is a strawman and may not be the actual keys used when this is implemented.

A nova user booting an instance with either the baremetal-gold or baremetal-gold-uefi flavor would land on node-1, because capabilities can still be passed down to ironic, and the resource_class on the node matches what is required by flavor. The baremetal-silver flavor would match node-2.

Alternatives

Keep doing the (host, node) thing long enough such that the nova team decides they want to remove our driver from nova, and subsequently remove the (host, node) tuple and break our driver horribly.

Data model impact

Adds a resource_class field to the nodes table. This will be a VARCHAR(80) because that matches novas table structure. There will be data migrations provided for this change. The objects API will also take this change, with a corresponding version bump. This will default to NULL.

State Machine Impact

None.

REST API impact

The new resource_class field will be exposed in the API, just like any other string field, with the same semantics.

Default policy for this field should be the same as driver, network_interface, etc.

Filtering and sorting on this field will be added.

There will be a microversion bump, as usual.

Client (CLI) impact

The client will be updated to add the field.

ironic CLI

The CLI will be updated to add the field.

openstack baremetal CLI

The CLI will be updated to add the field.

RPC API impact

Only the objects changes mentioned above.

Driver API impact

None.

Nova driver impact

Immediately, well pass the *resource_class* field back up to the resource tracker, so that nova can put these resources in the placement database in Newton. There will be a small patch that bumps the API version were using and passes the field back in the resource_dict. This will need a release note dictating the ironicclient version and available API version needed to run this code.

During Ocata, work will be done on the scheduler to use this for scheduling, however that is outside of the ironic driver.

Ramdisk impact

None.

Security impact

None.

Other end user impact

None.

Scalability impact

None.

Performance Impact

None.

Other deployer impact

After the deployment of the Newton version of nova, deployers will need to populate the resource_class field in ironic, and associate the flavors, before deploying the Ocata version of nova.

Developer impact

None.

Implementation

Assignee(s)

jroll

Work Items

- Add the field to the DB.
- Add the field to the objects model.
- Add the field to the API, with filtering and sorting.
- Doc updates for install guide and such.

Dependencies

None.

Testing

Unit tests should suffice here.

Upgrades and Backwards Compatibility

No direct impact, but its important to note that scheduling in the Ocata version of nova will still fall back to the old method, if resource_class is set to NULL for any node. Operators should have up until the P version of nova to populate this data.

Documentation Impact

Add the new field to the API reference.

Well also need to update the install guide and any other docs that talk about setting up nova with ironic, to make sure that deployers are setting this field when adding nodes. This will also need to be communicated extremely hard via release notes (and probably ops list emails).

References

[0] https://review.opendev.org/#/c/312696

Nova-compatible Serial Console

https://bugs.launchpad.net/ironic/+bug/1553083

This implements console interfaces using socat which provides nova- compatible serial console capability.

Problem description

Currently, ironics only console interface is based on shellinabox, which provides a stand-alone web console, and is not compatible with nova-serialproxy.

Proposed change

In order to address the problem of not having a serial console compatible with nova, this spec proposes using a command line tool `socat`[1] in conjunction with IPMI Serial-Over-Lan capabilities. `socat` is a command line based utility that establishes two bidirectional byte streams and transfers data between

[1] http://linux.die.net/man/1/socat

them. This application allows us to activate the ipmitool Serial-over-LAN process and redirect it through a TCP connection which can then be consumed by the `nova-serialproxy` service.

Each console (socat + ipmitool) will run on its own process on the same host that the ironic-conductor which is currently managing that node is running. `socat` will run first, then it will execute ipmitool when it has connections from `nova-serialproxy`, and will work like a `bridge` between them.

Start/stop of an ironic-conductor process

- When ironic-conductor starts, if console mode of the node is true, start socat also.
- When ironic-conductor stops, if console is started, stop it.
- About takeover work, were planning:
  - When an ironic-conductor stops, console session will be stopped due to security reason. In this case, if there are other ironic-conductors, they takeover nodes and enables their console session again.
  - When ironic-conductor starts, if there are console enabled nodes, the ironic-conductor starts their console.
- Start/stop console will be implemented with subprocess.Popen.
- About start/stop socat, were planning to implement a new console interface IPMISocatConsole and implement same methods as shellinabox classes. About this, discussed in:[2] .
- About reconnection, for example, in case of temporary network problem with using Horizon, Closed message will be shown. And socat itself supports session reconnect from client side, so that, when the network problem is resolved, users can try to reconnect.

Specify which of shellinabox or socat to use

Were planning to specify which driver to use shellinabox or socat by setting driver like `pxe_ipmitool_socat` or `agent_ipmitool_socat`. (Please see `Other deployer impact` section.)

Alternatives

Creating a new service ironic-xxx instead of adding a new ConsoleInterface to `ironic-conductor` . The upside of new service is that it can be scaled independently, and has no implications on conductor failover. However it will need its own HA model as well, and will be more work for developers (API, DB, driver,).

Data model impact

None

State Machine Impact

None

[2] https://review.opendev.org/#/c/293873/

REST API impact

The response body of GET /v1/nodes/<UUID>/states/console contains a JSON object like below:

```
{
  "console_enabled": true,
  "console_info": {
    "url": <url>,
    "type": <type>
  }
}
```

In case of using socat instead of shellinabox, <type> will be socat and <url> is like tcp://<host>:<port>.

Client (CLI) impact

None

RPC API impact

None

Driver API impact

None

Nova driver impact

get_serial_console() will be implemented in ironic driver of Nova. It returns a dictionary, similar to nova.virt.libvirt.driver.LibvirtDriver.get_serial_console(). No other impact for nova, and nova-serialproxy works well with the new one. And also, nova has agreed to the nova side of the work[3].

Ramdisk impact

None

Security impact

The connection between nova-serialproxy and socat is TCP based, like KVM. Socat supports OpenSSL connections, so we can improve the security in the future.

Other end user impact

None

Scalability impact

If a conductor can service 1000 nodes, and a process is created for a console to each node, but its the same scalability issue as shellinabox.

[3] https://blueprints.launchpad.net/nova/+spec/ironic-serial-console-support

Performance Impact

None

Other deployer impact

To use socat serial console, deployer needs to specify new driver. For example, to use PXE + IPMItool + socat, specify `pxe_ipmitool_socat`. To use IPA + IPMItool + socat, specify `agent_ipmitool_socat`. To use existing shellinabox console, deployer doesnt need to change anything. The new console interface `IPMISocatConsole` will be supported by two new drivers: `pxe_ipmitool_socat` and `agent_ipmitool_socat`. After `Driver composition reform`[4] is implemented, this feature will be available for a lot more drivers (or hardware types).

About configuration options, existing options `terminal_pid_dir`, `subprocess_checking_interval`, `subprocess_timeout` are available for socat in the same way as shellinabox. `terminal_cert_dir` is not used in the case of socat because SSL is not supported. `terminal` is not used in the case of socat because hard-coded `socat` is used in the code, and absolute path is not needed because its distro specific, in Ubuntu for example its `/usr/bin/socat`, but it might be different in other distros.

Developer impact

None

Implementation

Assignee(s)

Primary assignee:

- Akira Yoshiyama <akirayoshiyama@gmail.com>

Other contributors:

- Dao Cong Tien <tiendc@vn.fujitsu.com>
- Nguyen Tuong Thanh <thanhnt@vn.fujitsu.com>
- Cao Xuan Hoang<hoangcx@vn.fujitsu.com >
- Hironori Shiina <shiina.hironori@jp.fujitsu.com>
- Yuiko Takada Mori <y-mori@ti.jp.nec.com>

Work Items

- Implement `IPMISocatConsole` and `NativeIPMISocatConsole` class inherited from `base.ConsoleInterface`.

Dependencies

None

[4] https://review.opendev.org/#/c/188370/

Testing

Unit Testing will be added.

Upgrades and Backwards Compatibility

None

Documentation Impact

Add configuration description to the install guide.

References

Dynamic allocation of nodes for the OneView drivers

https://bugs.launchpad.net/ironic/+bug/1541096

This spec proposes a change in the way the OneView drivers allocate nodes. The new model will allocate resources in OneView only at boot time, avoiding that idle hardware in Ironic is still blocked from OneViews users perspective.

In this spec, the following terms will be used to refer to OneView resources:

- Server Hardware (SH): corresponds to a physical server. A Server Hardware can have only one Server Profile applied at a time.
- Server Profile (SP): representation of hardware configuration, such as network configuration, boot settings, firmware version, as well as features configurable through BIOS.
- Server Profile Template (SPT): template used to define a reference configuration of a set of Server Profiles.

Problem description

The current version of the OneView drivers consider what we call *pre-allocation* of nodes. This means that when a node is registered in Ironic, there must be a Server Profile applied to the Server Hardware represented by the given node.

In OneView, when a Server Hardware has a Server Profile applied to it, the understanding is that such hardware is being used.

The problem with *pre-allocation* is that even when a node is *available* in Ironic and, therefore, there is no instance associated to it, the resource in OneView is considered to be allocated. This means that no other OneView user can use and control this resource.

Therefore, in an environment shared by Ironic and OneView users, to simply allocate a pool of Server Hardware items to Ironic, for which we do not know when they are going to be used, will result in hardware reserved but not in use, which from many perspectives, specially for OneView users, is undesirable.

The ability to dynamically allocate nodes is currently missing from the set of OneView drivers. By adding this feature, the drivers will be able to have OneView resources allocated to Ironic only at boot time. Thus, this will enable the hardware pool to be actually shared among OneView and Ironic users.

Proposed change

In order to support dynamic allocation of nodes, some steps in the process of validating and deploying an OneView node need to be changed.

A node will be considered to be free for Ironic when (1) there is no Server Profile applied to the Server Hardware the node represents, or (2) there is a Server Profile applied, but this Server Profile is consistent with what is registered in Ironics node data structure. This means that the Server Profiles URI is equal to the value of the field *applied_server_profile_uri* in the *driver_info* namespace of the given node in Ironic.

Therefore, the following rules define the ownership of the node:

- SH without server_profile - Free
- SH.server_profile == node.driver_info.applied_server_profile_uri - Used by Ironic
- SH.server_profile != node.driver_info.applied_server_profile_uri - Used by OneView

When following this approach, a Server Profile is no longer required to validate a node. The *validate* methods of the *Power* and *Management* interfaces of the driver must be changed.

In the proposed model, if a node is not free for Ironic (i.e., it is being used by OneView) the validation will fail. However, as we expect nodes that are used by OneView to be moved to *manageable* and be cleaned before being made available again in Ironic, if the *target_provision_state* of a node is *manageable*, then this check will be skipped.

A node free for Ironic can be claimed by a OneView user at any moment. In that case, the node would still appear to be *available* in Ironic. To settle this case, a driver periodic task will be issued to detect nodes in use by OneView users and set them to *manageable* state and put into maintenance mode with a proper maintenance reason message. As soon as the node is freed by the OneView user, another driver periodic task will remove the maintenance mode and *provide* the node back into available. This is necessary to ensure the node is cleaned before being provided to an Ironic user.

If such node is scheduled for deployment prior to the execution of the periodic task, it will go to the *deploy failed* state and enter the cleaning process, that should fail since the node is in use by a OneView user. Nova scheduler will be able to pick a different node. A third periodic task will then be responsible to detect such failures and move the node back to *manageable* state and set maintenance mode until freed.

Some changes in the cleaning process must also be considered. In order to clean a node, a Server Profile needs to be assigned to it. Considering that, if a node has no Server Profile applied to it, a new temporary one will be created based on the Server Profile Template of this node, and applied to such hardware for cleaning purposes. If there is a Server Profile already applied to the node, it will be reused. After the cleaning is complete, in both cases, such Server Profile will be removed.

From a technical perspective, the *iscsi_pxe_oneview* and *agent_pxe_oneview* drivers will now implement:

- oneview.deploy.OneViewIscsiDeploy
- oneview.deploy.OneViewAgentDeploy

Both interfaces will override three methods:

- *prepare_cleaning*:

If the node is in use by OneView, an exception will be raised and the node goes to the *clean failed* state. Otherwise, cleaning steps will be performed and additional actions taken according to the following:

- **If there is no Server Profile assigned to the nodes Server Hardware:**

  - Server Profile will be created according to the Server Profile Template of the node;
  - Such Server Profile will be applied to the Server Hardware the node represents;
  - *applied_server_profile_uri* field will be added to the *driver_info* namespace of the node;

- *tear_down_cleaning*:

If the node is in use by Ironic, the following actions will be taken:

- *applied_server_profile_uri* field will be deleted from the *driver_info* namespace of the node;
- Server Profile will be removed from the Server Hardware the node represents;
- *prepare*:

If the node is in use by OneView, an exception will be raised and the node goes to a *deploy failed* state. Otherwise, additional actions will be taken as follows:

- Server Profile is applied to the Server Hardware represented by the node. The information required to perform such task will be recovered from the Server Profile Template indicated in the *server_profile_template_uri* inside the *properties/capabilities* namespace of the node.
- *applied_server_profile_uri* field will be added to the *driver_info* namespace of the node;

The interfaces will also implement three new driver periodic tasks:

- *_periodic_check_nodes_taken_by_oneview*:

This driver periodic task will check for nodes that were taken by OneView users while the node is in available state and set the node to maintenance mode with an appropriate maintenance reason message and move the node to *manageable* state.

- *_periodic_check_nodes_freed_by_oneview*:

This driver periodic task will be responsible to poll the nodes that are in maintenance mode and on *manageable* state to check if the Server Profile was removed, indicating that the node was freed by the OneView user. If so, itll *provide* the node, that will pass through the cleaning process and become available to be provisioned.

- *_periodic_check_nodes_taken_on_cleanfail*

This last driver periodic task will take care of nodes that would be caught on a race condition between OneView and a deploy by Ironic. In such cases, the validation will fail, throwing the node on *deploy fail* and, afterwards on *clean fail*. This task will set the node to maintenance mode with a proper reason message and move it to *manageable* state, from where the second task can rescue the node as soon as the Server Profile is removed.

A new configuration will be created on *[oneview]* section to allow operators to manage the interval in which the periodic tasks will run:

```
[oneview]
...
periodic_check_interval=300
```

Alternatives

Today, there is no other way to enable dynamic allocation of nodes with the OneView drivers.

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

ironic CLI

None

openstack baremetal CLI

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

N/A

Security impact

When a machine previously in use by an OneView user is released, its disks are not erased in the process since OneView does not perform such cleaning tasks. This means that once a node is released, some remaining data from its previous user can be available for Ironic. To prevent this leftover data from being exposed to Ironic users, the driver will move these machines to *manageable* state through its periodic tasks, where Ironic will require them to go through cleaning before being made *available* again. Note that if the cleaning feature is disabled in Ironic, OneView users are responsible for manually erase such disks prior to releasing the hardware.

Other end user impact

None

Scalability impact

None

Performance Impact

In most cases, applying a Server Profile in OneView takes less than 2 minutes.

In the few cases it can take longer, e.g. in firmware upgrades, the user must configure timeouts accordingly. Documentation will be provided on how to do it.

The performance impact of the periodic tasks on Ironic conductor will be proportional to the number of nodes being managed since theyll poll nodes in OneView to check if nodes were taken/returned by OneView users. Well minimize that impact by polling only nodes on specific known states as said previously on this spec. Yet, the *periodic_check_interval* can be adjusted according to OneView usage behavior and the number of nodes enrolled to reach an optimized performance of the conductor.

Other deployer impact

After this change is merged, the way the OneView drivers allocate nodes will be different. Deployers should be aware that:

- For existing nodes hosting instances, the *applied_server_profile_uri* field must be added to the *driver_info* namespace of the node.
- Nodes that had a Server Profile assigned to them but were not actually in use (according to the *pre-allocation* model), can have their Server Profiles removed and still be available in Ironic.

In order to ease this process, a migration tool will be provided. Check *Upgrades and Backwards Compatibility* for more details.

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
liliars

Other contributors:
sinval thiagop gabriel-bezerra marcusrafael nicodemos caiobo

Work Items

- Implement oneview.deploy.OneViewIscsiDeploy
- Implement oneview.deploy.OneViewAgentDeploy
- Override *prepare*, *prepare_cleaning*, *tear_down_cleaning* methods for both interfaces

- Implement the driver periodic tasks to deal with nodes taken by OneView users
- Write tests for such scenarios
- Document these changes

Dependencies

- python-oneviewclient version will be bumped

Testing

- Unit-tests will be implemented for the changes;
- The OneView third party CI will be used to provide a suitable test environment for tests involving a OneView appliance and specific hardware;

Upgrades and Backwards Compatibility

As said in *Other deployer impact* section of this spec, to migrate nodes that are in use with the *pre-allocation* model, one should add the field *applied_server_profile_uri* to the *driver_info* namespace of the node. Nodes that are on *available* state needs to have their Server Profiles removed. To ease this upgrade process on running deployments of Ironic, a migration tool will be provided and properly referenced in the drivers documentation.

Operators might have to increase the value of *workers_pool_size* and *periodic_max_workers* settings to allow to increase the greethread pool size and allow more than one parallel task to deal with nodes taken/returned on each periodic task as the pool of nodes on OneView increases.

Deprecation policy for *pre-allocation*:

- Newton:
  - Both pre-allocation and dynamic allocation will be supported
  - Flag to indicate whether dynamic allocation is enabled in driver_info
  - Driver defaults to pre-allocation in case the flag is missing
  - Script provided to ease migration process.
  - Deprecate *pre-allocation* feature in the driver code.
- Ocata:
  - Both pre-allocation and dynamic allocation will continue to be supported (due to deprecation process)
- P:
  - Drop support of *pre-allocation*
  - Flag ignored.

Documentation Impact

OneView drivers documentation will be updated accordingly. Some topics to be addressed are as follow:

- Definition of the new dynamic allocation model;
- Addition of fields in the node;

- Information regarding the migration process from the *pre-allocation* model;
- New configuration options and possible improvements;

References

OneView page
http://www8.hp.com/ie/en/business-solutions/converged-systems/oneview.html

OneView 2.0 REST API Reference
http://h17007.www1.hp.com/docs/enterprise/servers/oneview2.0/cic-rest/en/content/

python-oneviewclient
https://pypi.org/project/python-oneviewclient

5.11.3 6.0

Active Node Creation

https://bugs.launchpad.net/ironic/+bug/1526315

This spec is intended to allow a slightly more permissive interaction with the ironic API to allow an operator to migrate a hardware fleet to be managed by ironic.

Problem description

At present the ironic API explicitly sets the new state for nodes to the beginning step in the ironic workflow.

As part of hardware fleet lifecycle management, an operator expects to be able to migrate inventory and control systems for their hardware fleet utilizing their existing inventory data and allocation records. Ultimately this means that an imported host MAY already be allocated and unavailable for immediate allocation.

For an operator of multiple distinct OpenStack infrastructures, it is reasonable to permit an operator to migrate running baremetal hosts from one system to another system that are ultimately components in a larger infrastructure, while not immediately reprovisioning hardware.

Proposed change

Allow an API client to transition a node directly from the `MANAGEABLE` state to the `ACTIVE` state, bypassing actual deployment of the node.

- Creation of a new API provision_state verb of `adopt` that invokes the state transition of `ADOPTING`.
- Creation of a new machine state transition of `ADOPTING` which is valid only in the `MANAGEABLE` state and allows an operator to directly move a node to `active` state. This transition would be dependent upon a successful interface validation. Failure of this transition shall move nodes to an `ADOPTFAIL` which will allow users to be able to identify nodes that failed adoption.
- Creation of a new machine state of `ADOPTFAIL` which a machine is set to upon the `ADOPTING` transition failing. This state will allow a user to re-attempt `ADOPTING` via `adopt`, or attempt to transition the node to the `MANAGEABLE` state via `manage`. Additionally, the `ADOPTFAIL` state will be listed in the list of states that permit node deletion from the ironic database.
- API client update to provide CLI interface to invoke this feature.
- Creation of explicit documentation covering:

```
- Use cases of the feature while explicitly predicating that proper
  operation requires node validation to succeed.
- Explicitly detail that it is the operator's responsibility to
  define the node with all relevant appropriate configuration else
  the node could fail node state provision operations of ``rebuild``
  and ``delete``. Which would result in manual intervention being
  necessary.
- Explain the basic mechanics of the use of the adoption feature
  to users in order to help convey the importance of the correct
  information being populated.
```

Alternatives

The logical alternative is to remove restrictions in what an API client posts to allow the caller to explicitly state or invoke a node to be created in `ACTIVE` state. As the community desires full functionality of the node to exist upon being imported along with driver interface validation, the implementation appears to lend itself to be implemented as a state transition instead of pure API logic.

Alternatively, we could craft operator documentation to help assist operators in directly loading nodes into the ironic database, coupled with the caveats of doing so, and require that documentation is updated in lock-step with any database schema changes.

Data model impact

None

State Machine Impact

Implementation of a new state transition from `MANAGEABLE` state to `ACTIVE` state utilizing an intermediate state of `ADOPTING` which takes the following actions.

1. Triggering the conductor node take_over logic.
2. Upon completion the node state is set to `ACTIVE`.

Should a failure of take_over logic occur in the `ADOPTING` state, the node will be returned to `ADOPTFAIL` state from which a user will be able to retry the adoption operation or delete the node.

Addition of `ADOPTFAIL` into the `DELETE_ALLOWED_STATES` list.

REST API impact

Addition of a new state verb of `adopt` that triggers a transition to `ADOPTING` state. This verb will be unavailable for clients invoking an insufficient API version.

The API micro-version will need to be incremented as a result of this change.

Client (CLI) impact

Update of the ironicclient CLI to detail that `adopt` is a possible provision state option.

Update of the ironicclient micro-version.

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

N/A

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

Minimal API impact will exist for a user of this feature as the creation of nodes in `ACTIVE` state will require multiple calls with the API by any user attempting to leverage this feature.

Users performing bulk loads of hosts may find the multiple API calls somewhat problematic from the standpoint of multiple API calls to create, validate, and adopt a node, on top of API calls to poll the current state of the node before proceeding to the next step. Bulk loaders should also be congnizant of their configurations as they potentially could result in the conductors consuming disk space and network bandwidth if items need to be staged on the conductor to support the nodes normal operation.

Other deployer impact

Allows for an easier adoption by managers of pre-existing hardware fleets.

There is the potential that a operator could define a hardware fleet with bare minimal configuration to initially add the node to ironic. The result of which means that an operator could conceivably and inadvertently act upon a node when insufficient information is defined. This risk will be documented as part of the resulting documentation in order to help highlight the risk and help provide guidance on preventing such a possibility should a user be attempting to adopt an inventory that is already cloudy.

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
juliaashleykreger

Other contributors:
None

Work Items

- Conductor State Machine Changes
- API microversion and update and appropriate logic
- CLI node-set-provision-state option addition
- Documentation updates

Dependencies

None

Testing

Addition of unit tests as well as tempest tests to explicitly test the interface.

Upgrades and Backwards Compatibility

This feature will not be usable by older API clients via the API micro-version interface.

Documentation Impact

Documentation will need to be updated to represent this new feature.

References

None

5.12 Mitaka

5.12.1 5.1

Partition image support for agent driver

https://blueprints.launchpad.net/ironic/+spec/partition-image-support-for-agent-driver

This blueprint suggests to enhance agent driver to support deploy partition images with subsequent boot to happen over pxe or vmedia as specified by the driver.

Problem description

As of now agent driver support only whole disk images that can be deployed on the baremetal. With disk image based deploy, the subsequent boot will happen from the local hard drive. Ironic does not have control over the subsequent boots of the provisioned baremetal node.

Proposed change

- Agent driver validate the specified image type is of partition image(raw) by looking for kernel_id and ramdisk_id image properties.
- Send partition information(root, swap, etc) to the agent ramdisk through image_info.
- Upon receiving the partition information, agent ramdisk will work on the given os_install disk and copy the partition image in the root partition.
- Agent ramdisk sends back the root_uuid to the agent driver on the controller.
- Post deploy, agent driver prepares the config for subsequent boot, either using pxe or vmedia as defined by the driver. Both agent_ipmitool and agent_ilo driver should support deploy with partition images.
- Factor out the partitioning code from ironic into a different library and use it in both IPA and ironic code base.

Alternatives

We can use iscsi method to write partition image on to the target disk. We need agent ramdisk to support iscsi, similar to the ironic DIB element.

Data model impact

None.

State Machine Impact

None.

REST API impact

None.

Client (CLI) impact

None.

RPC API impact

None.

Driver API impact

None.

Nova driver impact

None.

Ramdisk impact

N/A

Security impact

None.

Other end user impact

Ability to deploy partition images on nodes managed by the agent.

Scalability impact

None.

Performance Impact

None.

Other deployer impact

None.

Developer impact

None.

Implementation

Assignee(s)

Primary assignee:
faizan-barmawer

Other contributors:
None

Work Items

- Factor out the partitioning code from ironic into a common library for both IPA and ironic code base.
    1. Move the disk partition code from ironic/common to an oslo incubator project as oslo.libironic

  - ironic/common/disk_partitioner.py
  - Some common disk related functions from ironic/drivers/modules/deploy_utils.py
  - Related test cases.

2. Use oslo.libironic in IPA

- Make necessary changes in agent driver common code, such as validate, deploy, etc.
- Make necessary changes in agent_ipmitool driver to generate correct pxe config for subsequent reboot.
- Make necessary changes in agent_ilo driver to generate iso for subsequent reboot.
- Make changes in IPA (agent ramdisk) to recognize the incoming image information and take appropriate action to deploy partition image on the disk.

Dependencies

Testing

- Unit testing with partition images with agent_ilo and agent_ipmitool drivers.
- Add specific agent driver test cases with partition images in tempest/devstack.

Upgrades and Backwards Compatibility

Documentation Impact

- Make changes to ironic install guide.

References

5.12.2 5.0

HTTP(S) proxy support for agent images downloading

https://bugs.launchpad.net/ironic/+bug/1526222

This adds support of proxy configuration for images downloading by agent.

Problem description

Currently Ironic Python Agent (IPA) is able to download images via direct HTTP(S) links, but it does not support proxy configuration. If IPA will support proxy configuration for image downloading user can place caching proxies in the same physical network segments as nodes for reducing owerall network traffic and deploying time. There are two different types of image sources when Ironic does deploy with IPA: Glance UUID and HTTP(S) URL. When HTTP(s) URLs are used so we can simply utilize HTTP(S) proxy configuration parameter, additional Ironic features are not needed. When we use Glance UUIDs there is a problem with Swift temporary URLs, because current time is used for temporary URLs calculation. In the proxy servers requests with query string parameters are cached separately for each unique query string, therefore if Swift temp URLs are used images can not be cached efficiently on the proxy server side.

Proposed change

Three new optional parameters: `image_http_proxy`, `image_https_proxy` and `image_no_proxy` will be added to agent deploy driver. First two parameters are strings with format PROTOCOL://PROXY_IP:PROXY_PORT. `image_no_proxy` is a list of comma-separated URLs that should be excluded from proxying. New behavior of agent deploy driver methods:

- get_properties() - returns description of new parameters.
- validate() - validate new parameter(s) (if present).
- continue_deploy() - add proxies and no_proxy keys in the image_info dict if parameter(s) present:

```
proxies = {'http': 'http://192.168.0.2:8080',
           'https': 'https://192.168.0.3:4444'}

no_proxy='192.168.1.5,10.0.0.3'
```

If proxies key is present IPA adds a parameter to requests.get() method. Requests library[0] supports no_proxy only as environment variable, not as a get() parameter. If no_proxy parameter is set agent should add it to Pythons os.environ before get() call.

Swift Temporary URL changes:

For caching proxies different URLs are mapped to different files in the cache. Therefore caching of Swift Temporary URLs for images should be implemented on the conductor. When a temporary URL for image is created agent driver stores it into the cache with UUID of Glance image as a key. Agent driver uses URL from cache for same UUIDs and checks expiration of temporary URLs. New integer config parameter `swift_temp_url_cachetime` will be added to `glance` group. If it greater than zero agent driver enables caching of URLs and use its value for new temp URL duration.

Notes about proxy service:

- Proxy should support HTTP/1.1 chunked transfer encoding.
- For SSL image URLs proxy should be configured for termination of SSL connection from client on the proxy side.
- Caching of large files should be enabled on the proxy.

Alternatives

None

Data model impact

None

State Machine Impact

None

[0] http://docs.python-requests.org/en/latest/user/advanced/#proxies

REST API impact

None

Client (CLI) impact

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

N/A

Security impact

Decrypting of HTTPS data on the proxy server side is not recommended for images which contain confidential information.

Other end user impact

None

Scalability impact

Proxy support for image downloading by agent can improve scalability (reduce network traffic and time of deploy) in proper configured environment.

Performance Impact

None

Other deployer impact

- New optional parameters will be added for agent deploy driver in the node.driver_info: `image_http_proxy`, `image_https_proxy`, `image_no_proxy`.
- A new config option `swift_temp_url_cachetime` will be added in `glance` group.
- Deployer must install and configure proxy service(s).

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
yuriyz

Work Items

- Implement proxy parameters for IPA deploy driver.
- Implement Swift Temporary URLs cache.
- Add unit tests.

Dependencies

None

Testing

Unittests will be added.

Upgrades and Backwards Compatibility

None

Documentation Impact

Usage of agents proxy configuration will be documented.

References

Switch periodic tasks to the Futurist library

https://bugs.launchpad.net/ironic/+bug/1526277

Futurist is a new Oslo library providing tools for writing asynchronous code. This spec suggests switching our periodic task implementation to Futurist to solve some long-standing problems.

Problem description

The main problem with our current implementation is that we run all periodic tasks in one thread. Any task that blocks for a while would block all other tasks from executing, and it happens pretty often with tasks checking power states via IPMI.

Switch to Futurist will allow executing all tasks in parallel.

Proposed change

- Modify conductor to use Futurist library instead of implementation from oslo incubator.

  Existing worker pool will be reused for periodic tasks. So, existing option `workers_pool_size` will set maximum number of tasks to run in parallel at every moment of time.

- Switch all use cases of `ironic.openstack.common.periodic_task` to Futurist decorators, and drop this module.
- Switch `ironic.drivers.base.driver_periodic_task` to using Futurist decorators internally and deprecate it.

Alternatives

- We could fix existing implementation. Thats not actually easier, as it requires essentially rewriting it.

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

RPC API impact

None

Driver API impact

- Old way of creating driver periodic tasks will be deprecated, drivers should eventually switch to using Futurist decorators.

Nova driver impact

None

Ramdisk impact

N/A

Security impact

None

Other end user impact

None

Scalability impact

- Overall positive impact on scalability expected, as every Ironic conductor will be able (at least theoretically) to manage more IPMI nodes.

Performance Impact

- A periodic tasks performance will no longer affect timing of other periodic tasks.

Other deployer impact

None

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
Dmitry Tantsur (irc: dtantsur, lp: divius)

Work Items

- Change conductor manager to use Futurist
- Modify `driver_periodic_task` to use Futurist internally

Dependencies

The only big dependency is Futurist itself. At the moment of writing it didnt see an official release yet, but is moving pretty fast and all required code already landed in git master.

Testing

Unit tests should already cover this functionality. Specific tests ensuring parallelization will be added.

Upgrades and Backwards Compatibility

None

Documentation Impact

Documentation of driver periodic tasks should be updated to mention Futurist instead of ad-hoc implementation.

References

- Futurist periodic task documentation

In-band RAID configuration using agent ramdisk

https://bugs.launchpad.net/ironic/+bug/1526398

This spec proposes to implement a RAID configuration interface using Ironic Python Agent (IPA). The drivers `agent_ipmitool`, `agent_pyghmi`, `agent_ssh`, `agent_vbox` and `agent_ilo` drivers will make use of this new implementation of `RAIDInterface`.

Problem description

Currently there is no way in Ironic to do RAID configuration for servers using in-band mechanism.

Proposed change

This spec proposes to implement in-band RAID configuration using IPA. It proposes to implement the `RAIDInterface` as mentioned in the parent spec [1]. The implementation will be named `AgentRAID`. The main job of the implementation will be to invoke the corresponding RAID operation methods on agent ramdisk. Interested vendors will implement these methods in Ironic Python Agent using hardware managers.

Following are the changes required:

- The following methods will be implemented as part of `AgentRAID`:
  - `create_configuration` - This will create the RAID configuration on the bare metal node. The following are the steps:
    - Uses `clean.execute_clean_step` command in Ironic Python Agent ramdisk to invoke the `create_configuration` step of `raid` interface.
  - `delete_configuration` - This will delete the RAID configuration on the bare metal node. The following are the steps:
    - Uses `clean.execute_clean_step` command in Ironic Python Agent ramdisk to invoke the `delete_configuration` step of `raid` interface.
- RAID configuration will be limited to zapping only at first by hardcoding its clean step priority to be 0. Well consider interaction with cleaning mechanism later. This allows us to make `target_raid_config` mandatory for the new clean steps.
- When the agent ramdisk is running an in-band clean step, the conductor gets the status of the last in-band clean step on every heartbeat. When an in-band clean step completes, the conductor resumes the cleaning and goes on to the next clean step if any. A new mechanism - the `agent_base_vendor.post_clean_step_hook` decorator, will be added. This allows a driver implementer to specify a function to be invoked after successful completion of an in-band clean step (and before the next clean step is started). The decorated function would take two arguments: the task and the command status (of the clean step) returned by the agent ramdisk.

For example:

```
@agent_base_vendor.post_clean_step_hook(
    interface='raid', step='create_configuration')
def _create_configuration_final(task, command):
```

- A method `agent._create_configuration_final` will be added as a post clean step hook for `raid.create_configuration`. This method will call `update_raid_info` with the actual RAID configuration returned from the agent ramdisk.
- A method `agent._delete_configuration_final` will be added as a post clean step hook for `raid.delete_configuration`. This will set `node.raid_config` to `None`. Note that `target_raid_config` will be left intact, and will be reused by future zapping calls.
- It is possible to have a hardware manager that does software RAID configuration, but it goes beyond the scope of this spec, as it requires RAID configuration to be run as a clean step after disk erasure.

Alternatives

Some bare metal servers do not support out-of-band RAID configuration. They support only in-band raid configuration. I dont see any other alternatives other than making use of a ramdisk to do this.

We could provide an option to enable RAID as part of cleaning. However, this will make `target_raid_config` mandatory for all nodes managed by a given conductor. Well have to reconsider it later.

Data model impact

None.

State Machine Impact

None.

REST API impact

None.

Client (CLI) impact

None.

RPC API impact

None.

Driver API impact

None.

Nova driver impact

None.

Ramdisk impact

N/A

Security impact

None.

Other end user impact

None.

Scalability impact

None.

Performance Impact

None.

Other deployer impact

None.

Developer impact

Other hardware vendors developing drivers for OpenStack can use Ironic Python Agent for in-band RAID configuration. They can add their own hardware manager implementing the method and get the RAID configuration done.

Implementation

Assignee(s)

rameshg87

Work Items

- Implement the mechanism for post clean step hook.
- Implement `AgentRAID`

Dependencies

- Implement Zapping States - https://review.opendev.org/140826

Testing

Unit tests will be added.

Upgrades and Backwards Compatibility

None.

Documentation Impact

None. Most of the RAID configuration details in Ironic are covered in the parent spec. If anything is required in addition, respective vendors making use of `AgentRAID` will need to document it.

References

[1] http://specs.openstack.org/openstack/ironic-specs/specs/approved/ironic-generic-raid-interface.html

New driver interface for RAID configuration

https://bugs.launchpad.net/ironic/+bug/1526400

The proposal presents the work required to create a new driver interface for RAID configuration. It also proposes a method to make RAID configuration available as part of zapping or cleaning.

Note

Even though RAID configuration fits into zapping, it can be used as part of cleaning as well. Zapping and cleaning follow a similar mechanism (zap step is a clean step with priority of 0). It makes sense to do RAID configuration as part of cleaning for software RAID (where secure disk erase will also erase the software RAID configuration on the disk). Its operators choice to decide whether RAID configuration should be part of zapping or cleaning and it will be configurable in the drivers implementing it.

Problem description

- There is no support in Ironic currently to do RAID configuration.
- A specific set of tasks for this requires a separate interface on the drivers. The new RAID interface will allow operators to specify RAID configuration for a node. Different drivers may provide the same interface to the operator for RAID configuration.

Proposed change

- After a node is enrolled and the basic hardware information is available, the operator can define a RAID configuration. This configuration will be applied during zapping or cleaning.
- The operator can convey the RAID configuration information to the Ironic driver through REST APIs or CLI as JSON data. The RAID configuration information will contain the properties for each logical disk and optionally hints to Ironic to find the desired backing physical disks for them.

  The properties can be split into 4 different types:

1. Mandatory properties - These properties must be specified for each logical disk and have no default values.
   - `size_gb` - Size (Integer) of the logical disk to be created in GiB. `MAX` may be specified if the logical disk should use all of the space available on the backing physical disks. This can be used only when backing physical disks are specified (see below).
   - `raid_level` - RAID level for the logical disk. Ironic will define the supported RAID levels as 0, 1, 2, 5, 6, 1+0, 5+0, 6+0. Drivers may override the values in the `get_logical_disk_properties` method in `RAIDInterface`.
2. Optional properties - These properties have default values and they may be overridden in the specification of any logical disk.
   - `volume_name` - Name of the volume. Should be unique within the Node. If not specified, volume name will be auto-generated.
   - `is_root_volume` - Set to `true` if this is the root volume. Can be used for only one of logical disk. The root device hint will be saved, if the driver is capable of retrieving it. This is `false` by default.
3. Backing physical disk hints - These hints are specified for each logical disk to let Ironic find the desired disks for RAID configuration. This is machine-independent information. This serves the use-case where the operator doesnt want to provide individual details for each bare metal node.
   - `share_physical_disks` - Set to `true` if this logical disk can share physical disks with other logical disks. If this is not provided, drivers will assume it as `false`.
   - `disk_type` - `hdd` or `ssd`. If this is not specified, disk type will not be considered as a criteria to find backing physical disks.
   - `interface_type` - `sata` or `scsi` or `sas`. If this is not specified, interface type will not be considered as a criteria to find backing physical disks.
   - `number_of_physical_disks` - Integer, number of disks to use for the logical disk. Defaulted to minimum number of disks required for the particular RAID level.

   The above mentioned backing physical disk hints are defined by Ironic and every driver has to implement them. The supported values and the default values for the above hints may be overridden by the driver using the `RAIDInterface.get_logical_disk_properties()` method.

   In addition to the above hints, drivers may define their own hints in the `get_logical_disk_properties` method. For more details, refer to the Driver API impact section. The possible use-cases for them might be:
   - Filter disks by particular vendors
   - Filter disks by models
   - Filter disks by firmware versions.
4. Backing physical disks - These are the actual machine-dependent information. This is suitable for environments where the operator wants to automate the selection of physical disks with a 3rd-party tool based on a wider range of attributes (eg. S.M.A.R.T. status, physical location).
   - `controller` - The name of the controller as read by the driver.
   - `physical_disks` - A list of physical disks to use as read by the driver.

> **Note**
>
> The values for these properties are hardware dependent.

> **Note**
>
> Only properties from Backing physical disk hints or Backing physical disks should be specified. If both are specified, they should be consistent with each other. If they are not consistent, then the raid configuration will fail (because the appropriate backing physical disks could not be found).

Some examples:

Example 1 (using backing physical disk hints):

```
{
  "logical_disks":
    [
      {
        "size_gb": 50,
        "raid_level": "1+0",
        "disk_type": "hdd",
        "interface_type": "sas",
        "volume_name": "root_volume",
        "is_root_volume": "true"
      },
      {
        "size_gb": 100,
        "number_of_physical_disks": 3,
        "raid_level": "5",
        "disk_type": "hdd",
        "interface_type": "sas"
        "volume_name": "data_volume"
      }
    ]
}
```

Example 2 (using backing physical disks):

```
{
  "logical_disks":
    [
      {
        "size_gb": 50,
        "raid_level": "1+0",
        "controller": "RAID.Integrated.1-1",
        "volume_name": "root_volume",
        "is_root_volume": "true"
        "physical_disks": [
                          "Disk.Bay.0:Encl.Int.0-1:RAID.Integrated.
```

(continues on next page)

(continued from previous page)

```
↪1-1",
                            "Disk.Bay.1:Encl.Int.0-1:RAID.Integrated.
↪1-1"
                        ]
        },
        {
          "size_gb": 100,
          "raid_level": "5",
          "controller": "RAID.Integrated.1-1",
          "volume_name": "data_volume"
          "physical_disks": [
                            "Disk.Bay.2:Encl.Int.0-1:RAID.Integrated.
↪1-1",
                            "Disk.Bay.3:Encl.Int.0-1:RAID.Integrated.
↪1-1",
                            "Disk.Bay.4:Encl.Int.0-1:RAID.Integrated.
↪1-1"
                        ]
        }
    ]
}
```

- The RAID configuration information is stored as JSON in `node.target_raid_config` field. Operator can use the REST API (or CLI) to put a new value here at any time, which is compared to `node.raid_config` during zapping and cleaning, and driver may apply changes only in those stages. Refer REST API Impact section for more details.
- New driver interface called `RAIDInterface` will be provided for RAID configuration for drivers. For more details, refer to the Driver API impact section.
- New methods `create_configuration` and `delete_configuration` in `RAIDInterface` will be available as part of zapping. The operator can choose to call them as part of zap steps. The corresponding zap steps will be `node.raid.create_configuration` and `node.raid.delete_configuration`.
- A new method `update_raid_info` will be available in ironic.common.raid. This method may be used by the drivers implementing RAID support to update the RAID information in the Node database. This will facilitate drivers to do the RAID configuration asynchronously. This method will do the following:
  - Set `node.raid_config` to the value returned by the driver.
  - The root device hint for the root volume will be updated in `node.properties` (as per root device hint) and the size of root volume will be updated in `node.properties.local_gb`. Its up to the driver to choose which root device hint it wants to specify. Furthermore, it isnt even necessary for the driver to choose any root_device_hint.
  - The RAID level of the root volume will be updated as `raid_level` in `node.properties.capabilities`.
- A new REST API will be created for retrieving the properties which may be specified as part of RAID configuration. For details, see the REST API Impact section below.
- REST API will be created to PUT RAID config, and a new REST resource added to retrieve the

requested and actual RAID config.

Alternatives

- Operator can change the RAID configuration manually whenever required after putting the node to MANAGEABLE state. But this has to be done for each node.

Data model impact

The following fields in the Node object will be updated:

- A new database field, `node.target_raid_config`, will store the pending RAID configuration to be applied during zapping or cleaning. This will be a JSON dictionary. This field will be read-only.
- A new database field, `node.raid_config`, will store the last applied RAID configuration. This will also contain the timestamp of when this configuration was applied. This will be a JSON dictionary. This field will be read-only.
- `node.properties.local_gb` will be updated after applying RAID configuration to the size of the root volume.
- `node.properties.root_device` will be updated with the root device hint returned by the driver as prescribed in the root device hint spec.
- A new capability `raid_level` will be added in `node.properties.capabilities`. This will contain the RAID level of the root volume.

State Machine Impact

None.

REST API impact

Two new REST API endpoints will be introduced as part of this change.

- To GET the RAID properties that can be defined and their possible values:

```
GET /drivers/<driver>/raid/logical_disk_properties
```

The operation will return the properties and a textual description of the possible values for each property:

```
{
 'raid_level': 'RAID level for the logical disk. Supported values are
                0, 1, 2, 5, 6, 1+0, 5+0, 6+0. Required.',
 'size_gb': 'Size of the logical disk in GiB. Required.',
 'disk_type': 'Disk Type. Supported values are `hdd` or `sdd`. Optional',
 .
 .
 .
 .
}
```

- To set the target RAID configuration, a user will:

```
PUT /v1/nodes/NNNN/states/raid
```

with a BODY containing the JSON description of the RAID config.

If accepted by the driver, this information will be stored in the `node.target_raid_config` field and exposed in the same manner as the power and provision states. In other words, it may be retrieved either within the detailed view of a `node`, or by either of the following:

```
GET /v1/nodes/NNNN
GET /v1/nodes/NNNN/states
```

Note

It might also make sense to have GET /v1/nodes/NNNN/states/raid, but for maintaining consistency with power and provision, we allow only GET /v1/nodes/NNNN and GET /v1/nodes/NNNN/states.

If the driver doesnt support RAID configuration, then both API calls will return HTTP 400 (Bad Request). Otherwise the API will return HTTP 200 (OK).

Client (CLI) impact

A new option will be available in Ironic CLI for getting the properties which may be specified as part of the RAID configuration:

```
$ ironic node-raid-logical-disk-properties <node-uuid>
```

A new method will be added to set the target RAID properties

RPC API impact

Two new RPC APIs will be created.

- `get_raid_logical_disk_properties` - This method will be called in `GET /drivers/<driver>/raid/logical_disk_properties`.
- `set_target_raid_config` - This method will be called in `PUT /v1/nodes/NNNN/states/raid`.

Driver API impact

A new `RAIDInterface` will be available for the drivers to allow them to implement RAID configuration. It will have the following methods:

- `create_configuration()` - The driver implementation of the method has to read the request RAID configuration from `node.target_raid_config` and create the RAID configuration on the bare metal. The driver implementations should throw error if `node.target_raid_config` is not set. The driver must ensure that `ironic.common.raid.update_raid_info()` is called at the end of the process, in order to update the nodes `raid_config`. The implementation detail is up to the driver depending on the synchronicity/asynchronicity of the operation.

  The `raid_config` will include the following:

- For each logical disk (in addition to the input passed):
  - `controller` - The name of the controller used for the logical disk as read by the driver.
  - `physical_disks` - A list containing the identifier for the physical disks used for the logical disk as read by the driver.
  - `root_device_hint` - A dictionary containing the root device hint to be used by Ironic to find the disk to which image is to be deployed. Its up to the driver to determine which root device hint it wants to provide.
- A list of all the physical disks on the system with the following details:
  - `controller` - RAID controller for the physical disk.
  - `id` - ID for the physical disk as read the driver
  - `disk_type` - `hdd` or `ssd`
  - `interface_type` - `sas` or `sata` or `scsi`
  - `size_gb`
  - `state` - State field states the current status of the physical disk. It can be one of:
    - `active` if disk is part of an array
    - `ready` if disk is ready to be part of a volume
    - `failed` if disk has encountered some error
    - `hotspare` if disk is hotspare and part of some array
    - `offline` if disk is not available for raid due to some other reason, but not failed
    - `non_raid` if disk is not part of raid and is directly visible

  The above details may be used for backing physical disk hints for later raid configurations.

> **Note**
>
> For a newly enrolled node or a node in which raid configuration was never done, the information about physical disks and controllers can be populated by hardware introspection. This is not in the scope of this spec.

The function definition will be as follows:

```
def create_configuration(task, create_root_volume=False,
                         create_nonroot_volumes=False):
    """Create RAID configuration on the node.

    This method creates the RAID configuration as read from
    node.target_raid_config.  This method
    by default will create all logical disks.

    :param task: TaskManager object containing the node.
    :param create_root_volume: Setting this to False indicates
        not to create root volume that is specified in the node's
```

(continues on next page)

(continued from previous page)

```
        target_raid_config. Default value is True.
    :param create_nonroot_volumes: Setting this to False indicates
        not to create non-root volumes (all except the root volume) in
        the node's target_raid_config.  Default value is True.
    :returns: states.CLEANWAIT if RAID configuration is in progress
        asynchronously or None if it is complete.
    """
```

- `delete_configuration()` - To delete the RAID configuration.

  The function definition will be as follows:

```
def delete_configuration(task):
    """Delete RAID configuration on the node.

    :param task: TaskManager object containing the node.
    :returns: states.CLEANWAIT if deletion is in progress
        asynchronously or None if it is complete.
    """
```

- `validate()` - To validate a RAID configuration. This will be called while validating the driver interfaces. This will read the target RAID configuration from node.properties.target_raid_config and call `validate_raid_config` to validate target RAID configuration.

  The function definition will be as follows:

```
def validate(task):
    """Validates the RAID interface.

    :param task: TaskManager object containing the node.
    :raises: InvalidParameterValue, if RAID configuration is invalid.
    :raises: MissingParameterValue, if RAID configuration has some
        missing parameters.
    """
```

- `validate_raid_config()` - To validate target RAID configuration. This will be called during the RPC call `set_target_raid_config()` to validate target RAID configuration. It will also be called during `validate()`.

  The function definition will be as follows:

```
def validate_raid_config(task, raid_config):
    """Validates the given RAID configuration.

    :param task: TaskManager object containing the node.
    :param raid_config: The target RAID config to validate.
    :raises: InvalidParameterValue, if RAID configuration is invalid.
    """
```

- `get_logical_disk_properties()` - To get the RAID properties that are defined by the driver.

  The function definition will be as follows:

```
def get_logical_disk_properties():
    """Gets the RAID properties defined by the driver.

    :returns: A dictionary of properties and a textual description.
    """
```

After performing the RAID configuration (create or delete), the drivers may call `ironic.common.raid.update_raid_info()` with the `raid_config`. The details about the method has been described above. The definition of the method will look like below:

```
def update_raid_info(node, raid_config):
    "Updates the necessary fields of the node after RAID configuration.

    This method updates the current RAID configuration in
    node.properties.raid_config.  If root device hint was passed,
    it will update node.properties.local_gb, node.properties.root_device_hint
    and node.properties.capabilities['raid_level'].

    :param node: a node object
    :param raid_config: The current RAID configuration on the bare metal
        node.
    """
```

Nova driver impact

None.

Ramdisk impact

N/A

Security impact

None.

Other end user impact

Users from Nova may choose the desired RAID level for the root volume by using compute capabilities. For example:

```
nova flavor-key ironic-test set capabilities:raid_level="1+0"
```

Scalability impact

None.

Performance Impact

RAID configuration may extend the time required for zapping or cleaning on the nodes, but this is important for performance and reliability reasons.

Other deployer impact

Operator can make use of `node.raid.create_configuration` and `node.raid.delete_configuration` as zap or clean tasks for doing RAID management.

Developer impact

Developers may implement the `RAIDInterface` for respective drivers.

Implementation

Assignee(s)

Primary assignee:
rameshg87

Other contributors:
ifarkas

Work Items

- Create REST API endpoints for RAID configuration.
- Create `RAIDInterface` and create a fake implementation of `RAIDInterface`.
- Implement `update_raid_info` in ironic.common.raid.
- Implement Ironic CLI changes.
- Write unit tests.

Dependencies

- Root device hints - http://specs.openstack.org/openstack/ironic-specs/specs/kilo/root-device-hints.html
- Zapping of nodes - https://review.opendev.org/#/c/140826/

Testing

- Unit tests will be added for the code. A fake implementation of the `RAIDInterface` will be provided for testing purpose and this can be run as part of zapping.
- Tempest API coverage will be added, using the fake driver above.
- Each driver is responsible for providing the third party CI for testing the RAID configuration.

Upgrades and Backwards Compatibility

None.

Documentation Impact

Documentation will be provided on how to configure a node for RAID.

References

Other references:

- New Ironic provisioning state machine: http://specs.openstack.org/openstack/ironic-specs/specs/kilo/new-ironic-state-machine.html
- Support Zapping of Nodes: https://review.opendev.org/#/c/140826/

Manual cleaning

https://bugs.launchpad.net/ironic/+bug/1526290

Manual cleaning (as opposed to automated cleaning) encompasses all long running, manual, destructive tasks an operator may want to perform either between workloads, or before the first workload has been assigned to a node.

This feature had previously been called Zapping and this specification copies a lot of the zapping specification. (Thank you Josh Gachnang!)

Problem description

Automated cleaning has been available in ironic since the kilo cycle. It lets operators choose which clean steps are automatically done prior to the first time a node is deployed and each time after a node is released.

However, operators may want certain operations or tasks to only run on demand, rather than in every clean cycle. Things like firmware updates, setting up new RAID levels, or burning in nodes often need to be done before a user is given a server, but take too long to reasonably do at deploy time.

Many of the above tasks could provide useful scheduling hints to nova once hardware capabilities are introduced. Operators could use these scheduling hints to create flavors, such as a nova compute flavor that requires a node with RAID 1 for extra durability.

Proposed change

Instead of adding new ZAP* states to the state machine to distinguish between manual and automated cleaning, the existing CLEAN* states and cleaning mechanism will be reused for both automated and manual cleaning. The main differences will be:

- manual cleaning can only be initiated when a node is in the MANAGEABLE state. Once the manual cleaning is finished, the node will be put in the MANAGEABLE state again.
- operators will be able to initiate a manual clean via the modified API to set the nodess provision state. Details are described in the *PUT /states/provision* section below.
- A manual clean step might need some arguments to be specified. (This might be useful for future automated steps too.) To support this, the ironic.drivers.base.clean_step decorator will be modified to accept a list of arguments. (Default is None.) Each argument is a dictionary with:

  - name: <name of argument>
  - description: <description>. This should include possible values.
  - required: Boolean. True if this argument is required it must be specified in the manual clean request; false if it is optional.

- add clean steps to drivers that will only be used by manual cleaning. The mechanism for doing this exists already. Driver implementers only need to use the @clean_step decorator with a default cleaning priority of 0. This will ensure the step isnt run as part of the automated cleaning. The implementer can specify whether the step is abortable, and should also include any arguments that can be passed to the clean step.
- operators will be able to get a list of possible steps via an API. The *GET /cleaning/steps* section below provides more information.
- similar to executing automated clean steps, when the conductor attempts to execute a manual clean step, it will call execute_clean_step() on the driver responsible for that clean step.
- to avoid confusion, the clean_nodes config will be renamed to automated_clean_enable since it only pertains to automated cleaning. The deprecation and deletion of the clean_nodes config will follow ironics normal deprecation process.

Alternatives

- We could make manual clean steps and automated clean steps mutually exclusive with separate APIs and terminology and mechanisms to use, but conceptually, since they are all clean steps it is less confusing to provide a similar mechanism for both.
- We could have called manual clean something else like zap to avoid having to distinguish between manual and automated cleaning, but it seems more confusing to describe the differences between zap and clean and that confusion and complexity is apparent when trying to implement it that way.

Data model impact

None.

State Machine Impact

This:

- removes all mention of zap and the ZAP* states from the proposed state machine
- adds two new transitions:
  - MANAGEABLE -> CLEANING via clean verb, to start manual cleaning
  - CLEANING -> MANAGEABLE via manage verb, to end a successful manual clean

REST API impact

PUT /v1/nodes/<node_ident>/states/provision

This API will allow users to put a node directly into CLEANING provision state from MANAGEABLE state via target: clean. The PUT will also require the argument clean_steps to be specified. This is an ordered list of clean steps, with a clean step being represented as a dictionary encoded as JSON.

As an example:

```
'clean_steps': [{
    'interface': 'raid'
    'step': 'create_configuration',
    'args': {'create_nonroot_volumes': False, // optional keyword argument
             ... }              // more keyword arguments (if applicable)
  },
  {
    'interface': 'deploy'
    'step': 'erase_devices'
  }
]
```

In the above example, the drivers RAID interface would configure hardware RAID without non-root volumes, and then all devices would be erased (in that order).

A clean step is represented by a dictionary (JSON), in the form:

```
{
    'interface': <interface>,
    'step': <name of clean step>,
    'args': {<arg1>: <value1>, ..., <argn>: <valuen>}
}
```

The interface and step keys are required for all steps. If a step takes additional keyword arguments, the args key may be specified. It is a dictionary of keyword arguments, with each keyword-argument entry being <name>: <value>.

If any step is missing a required keyword argument, no manual cleaning will be performed and the node will be put in CLEANFAIL provision state with an appropriate error message.

If, during the cleaning process, a clean step determines that it has incorrect keyword arguments, all earlier steps will be performed and then the node will be put in CLEANFAIL provision state with an appropriate error message.

A new API version is needed to support this.

GET /nodes/<node_ident>/cleaning/steps

We had planned on having an API endpoint to allow operators to see the clean steps for an automated cleaning. That proposed API had been GET /nodes/<node_ident>/cleaning/clean_steps, but it hasnt been implemented yet.

With the introduction of manual cleaning, instead of GET /nodes/<node_ident>/cleaning/clean_steps, this proposes replacing that with the API endpoint GET /nodes/<node_ident>/cleaning/steps. By default, it will return all available clean steps (with priorities of zero and non-zero), for both manual and automated cleaning.

An optional field min_priority can be specified to filter for clean steps with priorities equal to or above the specified minimum value. For example, to only get clean steps for automated cleaning (not manual):

```
GET http://127.0.0.1:6385/v1/nodes/my-awesome-node/cleaning/steps?min_
↪priority=1
```

The response to this request would be a list of clean steps sorted in decreasing priorities, formatted as follows:

```
[{
  // 'interface': is one of 'power', 'management', 'deploy', 'raid'.
  // 'step': is an opaque identifier used by the driver. Could be a driver
  //         function name or some function in the agent.
  // 'priority': is the priority used for determining when to execute
  //             the step; larger values have higher priority.
  // 'abortable': True if cleaning can be aborted during execution of this
  //              step; False otherwise.
  'interface': 'interface',
  'step': 'step',
  'priority': Integer,
  'abortable': Boolean

  // 'args': a list of keyword arguments that may be included in the
  //         'PUT /v1/nodes/NNNN/states/provision' request when doing
  //         a manual clean. An argument is a dictionary with:
  //           - 'name': <name of argument>
  //           - 'description': <description>
  //           - 'required': Boolean. True if required; false if optional
  'args': []
 },
 ... more steps ...
]
```

An example with a single step:

```
[{
  'interface': 'raid',
  'step': 'create_configuration',
  'args': [{'name':'create_root_volume',
            'description':'Set to True (the default) to create root volume
                           specified in the node's target_raid_config. False
                           prevents the root volume from being created.',
            'required':False},
           {'name':'create_nonroot_volumes',
            'description':'Set to True (the default) to create non-root
                           volumes that may be specified in the node's
                           target_raid_config. False prevents non-root
                           volumes from being created.',
            'required':False}]
  'priority': 0,
  'abortable': True
}]
```

If the driver interface cannot synchronously get the list of clean steps, for example, because a remote agent is used to determine available clean steps, then the driver MUST cache the list of clean steps from the most recent execution of said agent and return that. In the absence of such data, the driver MAY raise an error, which should be translated by the API service into:

- an HTTP 202
- a new (we created this) HTTP header Retry-Request-After, indicating to the client how long in seconds the client should wait to retry. A -1 indicates that it is unknown how long to wait. This might happen for example when the request is made when a node is in ENROLL state. At this point it is unknown when the remote agent will be available on the node for querying.
- a body with a message indicating that the data are not available yet.

If the driver interface can synchronously return the clean steps without relying on the hardware or a remote agent, it SHOULD do so, though it MAY also rely on the aforementioned caching mechanism.

A new API version is needed to support this.

Client (CLI) impact

ironic node-set-provision-state

A new argument called clean-steps will be added to the node-set-provision-state CLI. Its value is a JSON file which is read and the contents passed to the API. Thus, the file has the same format as what is passed to the API for clean steps.

If the input file is specified as -, the CLI will read in from stdin, to allow piping in the clean steps. Using - to signify stdin is common in Unix utilities.

The clean-steps argument is required if the requested provision state target/verb is clean. Otherwise, specifying it is considered an error.

ironic node-get-clean-steps

A new node-get-clean-steps API will be added as follows:

```
ironic node-get-clean-steps [--min_priority <priority>] <node>

<node>: name or UUID of the node
--min-priority <priority>: optional minimum priority; default is 0 for all␣
↪clean steps
```

If successful, it will return a list of clean steps. If the response from the corresponding REST API request is an HTTP 202, it will return the message from that response body (that the data are not available) along with a suggestion to retry the request again.

RPC API impact

Add do_node_clean() (as a call()) to the RPC API and bump the RPC API version.

Driver API impact

None

Nova driver impact

None

Ramdisk impact

N/A

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

None

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
rloo (taking over from JoshNang who has left ironic)

Other contributors:
JoshNang (who started this)

Work Items

- Make the changes (as described above) to the state machine
- Bump API microversion to allow manual cleaning and implement the changes to PUT /v1/nodes/(node_ident)/states/provision API (as described above)
- Modify the cleaning flow to allow manual cleaning
- Change execute_clean_steps and get_clean_steps in any asynchronous driver to cache clean steps and return cached clean steps whenever possible.
- Allow APIs to return a Retry-Request-After HTTP header and empty response, in response to a certain exception from drivers.

Dependencies

- get_clean_steps API: https://review.opendev.org/#/c/159322

Testing

- Drivers implementing manual cleaning will be expected to test their added features.

Upgrades and Backwards Compatibility

None

Documentation Impact

The documentation will be updated to describe or clarify automated cleaning and manual cleaning and how to configure ironic to do one or both of them:

- http://docs.openstack.org/developer/ironic/deploy/install-guide.html
- http://docs.openstack.org/developer/ironic/deploy/cleaning.html
- http://docs.openstack.org/developer/ironic/webapi/v1.html will be updated to reflect the API version that supports manual cleaning

References

Automated cleaning specification: http://specs.openstack.org/openstack/ironic-specs/specs/kilo-implemented/implement-cleaning-states.html

State machine specification: http://specs.openstack.org/openstack/ironic-specs/specs/kilo-implemented/new-ironic-state-machine.html

Zapping related patches:

- Launchpad blueprint: https://blueprints.launchpad.net/ironic/+spec/implement-zapping-states
- **specification patches:**
  - https://review.opendev.org/#/c/185122/
  - https://review.opendev.org/#/c/209207/
- **code patches:**
  - https://review.opendev.org/#/c/221949/
  - https://review.opendev.org/#/c/221989/
  - https://review.opendev.org/#/c/223295/
  - https://review.opendev.org/#/c/223311/

5.12.3 4.3

In-band cleaning support in iSCSI deploy drivers

https://bugs.launchpad.net/ironic/+bug/1526405

Current deploy drivers that make use of iSCSI dont support in-band cleaning. We need to make these drivers support in-band cleaning operations as well.

Problem description

Drivers that do iSCSI based deployment dont support in-band cleaning today. Hence in-band cleaning steps like disk erase, in-band RAID configuration, etc cannot be performed on these nodes which are registered with these drivers.

The following drivers (omitting the testing drivers) do iSCSI based deployment today:

- **pxe_{ipmitool,ipminative,seamicro,iboot,ilo,**
  drac,snmp,irmc,amt,msftocs,ucs,wol}
- iscsi_ilo
- iscsi_irmc

Proposed change

- Deprecate the opt `CONF.agent.agent_erase_devices_priority` and move that to `CONF.deploy.agent_erase_devices_priority`.
- Add the following methods to iscsi_deploy.ISCSIDeploy (these methods will do the same things as what AgentDeploy does today)
  - `prepare_cleaning` - This method will create the Neutron cleaning ports for each of the Ironic ports and will call `self.boot` to prepare for booting the ramdisk and return `states.CLEANWAIT`. The method definition will be as follows:

```
def prepare_cleaning(self, task):
    """Boot into the agent to prepare for cleaning.

    :param task: a TaskManager object containing the node
    :raises NodeCleaningFailure: if the previous cleaning ports␣
↪cannot
        be removed or if new cleaning ports cannot be created
    :returns: states.CLEANWAIT to signify an asynchronous prepare
    """
```

  - `tear_down_cleaning` - This method will delete the cleaning ports created for the Ironic ports and will call `self.boot` to clean up the ramdisk boot. It will return None. The method definition will be as follows:

```
def tear_down_cleaning(self, task):
    """Cleans up the environment after cleaning.

    :param task: a TaskManager object containing the node
    :raises NodeCleaningFailure: if the cleaning ports cannot be
        removed
    """
```

  - `execute_clean_step` - This method will call `deploy_utils.agent_execute_clean_step`. The method definition will be as follows:

```
def execute_clean_step(self, task, step):
    """Execute a clean step asynchronously on the agent.
```

(continues on next page)

(continued from previous page)

```
    :param task: a TaskManager object containing the node
    :param step: a clean step dictionary to execute
    :raises: NodeCleaningFailure if the agent does not return a␣
↪command
        status
    :returns: states.CLEANWAIT to signify the step will be completed
        async
    """
```

- `get_clean_steps` - This method will call `deploy_utils.agent_get_clean_steps` to get the cleaning steps from the agent ramdisk. It will also reassign the cleaning priority to disk erase.

  This will return an empty list if bash ramdisk is used. It will be detected by checking if `agent_url` is present in nodes driver_internal_info.

  The method definition will be as follows:

```
def get_clean_steps(self, task):
    """Get the list of clean steps from the agent.

    :param task: a TaskManager object containing the node
    :returns: A list of clean step dictionaries. Returns an
        empty list if bash ramdisk is used.
"""
```

- For deployers who have been using DIB ramdisk, the node will be stuck i n `states.CLEANWAIT` when they try to do cleaning. This is because DIB ramdisk doesnt heartbeat like agent ramdisk. Hence, such deployers might face the issue with nodes moving to `states.CLEANFAIL` as node enters cleaning. To overcome this problem, the following will be done:
  - Send the bash ramdisk parameters (deploy_key, iscsi_target_iqn, etc) while booting the deploy ramdisk for cleaning. It will enable bash ramdisk to invoke pass_deploy_info vendor passthru.
  - If node is in CLEANWAIT in pass_deploy_info vendor passthru, then we set the clean steps for the node and ask conductor to resume cleaning.
  - We also skip validation for pass_deploy_info vendor passthru if node is in CLEANWAIT state.

Alternatives

None.

Data model impact

None.

State Machine Impact

None.

REST API impact

None.

Client (CLI) impact

None.

RPC API impact

None.

Driver API impact

None.

Nova driver impact

None.

Ramdisk impact

N/A

Security impact

Drivers using `iscsi_deploy.ISCSIDeploy` will do in-band disk erase which will be a security benefit for tenants.

Other end user impact

None.

Scalability impact

None.

Performance Impact

None.

Other deployer impact

None.

Developer impact

None.

Implementation

Assignee(s)

Primary assignee:
rameshg87

Work Items

- Add new methods the `iscsi_deploy.ISCSIDeploy` for in-band cleaning.
- Modify pass_deploy_info to make it ready when it is invoked during cleaning.

Dependencies

- Completion of work for deploy-boot interface separation [2] to enable in-band cleaning for all drivers.

Testing

Unit tests will be added.

Upgrades and Backwards Compatibility

None.

Documentation Impact

The new CONF option and its impact will be documented.

References

[1] http://specs.openstack.org/openstack/ironic-specs/specs/approved/deprecate-bash-ramdisk.html [2] https://blueprints.launchpad.net/ironic/+spec/new-boot-interface

New driver in Ironic for OneView

https://bugs.launchpad.net/ironic/+bug/1526406

This spec proposes adding a new driver that supports deployment of servers managed by OneView. OneView is an Integrated Infrastructure Systems Management Software developed by HP.

In this spec, *Server Hardware* is the label used on OneView to denote a physical server.

Problem description

Currently Ironic does not have integration with any Infrastructure Management System. Nowadays, being able to use hardware from these systems inventory to provision a baremetal instance is a manual/time consuming task that would require pre-configuration of each server. OneView eases this configuration workload.

This spec proposes a new Ironic OneView driver that will promote integration with the HP OneView Management System. The proposed driver will provide automatic inventory management with OneView, allowing Ironic to borrow non-dedicated servers from OneViewss inventory to provision baremetal instances with minimal common pre-configuration, set through OneViews *Server Profile Templates* (SPT).

In order to use a Server Hardware managed by OneView, one needs to assign it a *Server Profile*, prior to the enrollment of the node, in order to configure the hardware to be used (select/update firmware, enable NICs, setup the network connections on them, BIOS/UEFI settings, initialize storage and so on). The cloud administrator can take advantage of such a resource to configure the servers NIC to use the Ironic provision network on the fly as needed, along with other options that could allow optimal performance.

Proposed change

This spec proposes the *pxe_oneview* and *agent_oneview* drivers, implementing the Power, Management and, in the case of the Agent driver, Vendor interfaces.

The driver uses *python-oneviewclient* in order to handle the communication between the driver and OneView for, e.g., getting information about a resource, turning a server hardware on and off, and handling the configuration of a server profile.

Server profiles are based on SPTs, which have information to configure the hardware NIC to join a flat network, initialize the server storage and other configuration options used by Ironic like boot_type and boot_order. The SPT can also hold specific configuration options to improve the hardware performance for Ironic, like Advanced Memory Protection, USB boot, enable Virtualization Technology and Hyper-Threading, change the thermal configuration and so on. Based on this premises, to be enrolled, the node MUST have the following parameters:

- **driver_info**
  - server_hardware_uri: URI of the Server Hardware on OneView
  - server_profile_template_uri: URI for the Server Profile Template used to create the Server Profile of the node. This will be used on the future to change the Server Profile of the node on a zapping task.
- **properties/capabilities**
  - server_hardware_type_uri: URI for the Server Hardware Type on OneView, for scheduling purposes if one wants to deploy on specific hardware determined on the flavor.
  - enclosure_group_uri: URI for the Enclosure Group on OneView, for scheduling purposes if one wants to deploy on specific enclosure determined on the flavor.

The driver implements:

- oneview.power.OneViewPower
- oneview.management.OneViewManagement
- oneview.vendor.AgentVendorInterface

Power Interface:
The *_*oneview* drivers Power Interface controls and synchronizes the power state of the nodes using OneViews REST API. The validate() method on this interface will check the required parameters and if the node already has a server profile associated.

Management Interface:
The *_*oneview* drivers Management Interface allows the user to get and set the boot-order of a server hardware by modifying the server profile assigned to the server hardware. If no server

profile is assigned yet to an instance, an exception will be thrown since the boot order of a server managed by OneView can only be modified through a server profile. The validate() method on this interface will also check the required parameters and if the node already has a server profile associated.

Agent Vendor Interface:
The *agent_oneview* interface modifies the way *reboot_to_instance* method sets the boot device since OneView doesnt allow such a change with the machine powered on.

This driver reuses PXEBoot for boot and ISCSIDeploy/AgentDeploy for deploy.

To be deployed using the **_oneview* driver, the nodes Server Profile MUST be applied to the server hardware the node represents. This Server Profile MUST connect the 1st NIC of the node to Ironics provision network.

Alternatives

We could use the already existing drivers (such as pxe_ipmitool, pxe_ilo, iscsi_ilo or even agent_ilo) to launch instances managed by OneView. But then: - We would lose the capability to manage these instances through OneView; - If the node is being managed by OneView, without a Server Profile, and deployed with other driver, another user can claim it by applying a Server Profile and thus Ironic would lose control of the server; - Without using OneView, the task of maintaining configuration consistency between the Server Hardware items is manual, boring and time consuming.

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

N/A

Security impact

The connection with OneView is by default secure using TLS with certificate authentication, but the user can allow insecure connections by setting to True the allow_insecure_connections field in the configuration file.

Other end user impact

None

Scalability impact

The driver gets some data using *python-oneviewclient* through OneViews REST API which is an external service. The calls are simple, but considering a large amount of Server Hardware items a small increase in network traffic can happen.

Performance Impact

None

Other deployer impact

The following parameters are required in the newly created [oneview] section on ironic.conf:

- manager_url: OneView Manager url
- username: User account with admin/server-profile access privilege in OneView
- password: User account password in OneView
- allow_insecure_connections: Allow connections to OneView without a certificate signed by a trusted CA. Its default value is False.
- tls_cacert_file: The path to the certificate of a trusted CA to be used to verify the OneView certificate when insecure connections are not allowed
- max_polling_attempts: Max connection attempts to check changes on OneView

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
thiagop

Other contributors:
albertoffb caiobo diegolp liliars sinval afaranha

Work Items

- Implement new *iscsi_pxe_oneview* and *agent_pxe_oneview* drivers.
- Implement unit-test cases for **_oneview* driver.
- Write configuration documents.

Dependencies

- The driver requires python-oneviewclient package.

Testing

Unit-tests will be implemented for the new drivers. A third party CI will be used in the future to provide a suitable test environment for tests involving an OneView appliance.

Upgrades and Backwards Compatibility

None

Documentation Impact

The required parameters on the node and *[oneview]* section of *ironic.conf* will be included in the documentation to instruct operators how to use Ironic with OneView.

References

OneView Page
http://www8.hp.com/ie/en/business-solutions/converged-systems/oneview.html

OneView REST API Reference
http://h17007.www1.hp.com/docs/enterprise/servers/oneviewhelp/oneviewRESTAPI/content/images/api/index.html

python-oneviewclient
https://pypi.org/project/python-oneviewclient

Ceph Object Gateway Temp URL support

https://bugs.launchpad.net/ironic/+bug/1526395

This adds support of Ceph Object Gateway (RADOS Gateway) temporary URL format.

Problem description

Ceph project is a powerful distributed storage system. It contains object store with OpenStack Swift compatible API. Glance image service can use Ceph storage via RADOS Gateway Swift API. Ironic does not currently support deploy configuration with Glance and RADOS Gateway. The reason is different format of temporary URL. First part of temporary URL for RADOS Gateway with Glance frontend is endpoint_url/api_version/container/object_id, where

- endpoint_url contains scheme, hostname, optional port and mandatory /swift suffix.
- api_version is v1 currently.

- container is the name of Glance container.
- object_id is Glance object id.

Calculation of parameters temp_url_sig and temp_url_expires is mostly the same as in Swift, so full URL looks like

https://radosgw.my.host/swift/v1/glance/22aee8e5-cba3-4554-92c4aadde5e38f28?
temp_url_sig=e75d1d6facb53d795547b1fc60eca4e8836bd503 &temp_url_expires=1443518434

temp_url_sig calculation should not use /swift in the path.

OpenStack Swift temporary URL contains extra account parameter, its account that Glance uses to communicate with Swift. swift_account parameter is mandatory for Ironic.

Note

Do not use Python code as reference from http://docs.ceph.com/docs/master/radosgw/swift/tempurl/ it does not create valid URLs, for Firefly release at least.

Proposed change

A new configuration parameter `temp_url_endpoint_type` will be added to the `glance` group. It can be set to values swift or radosgw, swift is default. Code of image service in Ironic will be changed for supporting both of endpoints (parameters set, mandatory suffix for RADOS Gateway).

Alternatives

None

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

N/A

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

- A new config option `temp_url_endpoint_type` will be added in `glance` group.
- Deployer should configure Glance with RADOS Gateway beckend (via Swift API) and Ceph storage.

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
yuriyz

Work Items

- Implement Rados GW support.
- Add unit tests.

Dependencies

None

Testing

Unittests will be added.

Upgrades and Backwards Compatibility

None

Documentation Impact

Usage of Ironic with Rados Gateway as Glance backend will be documented.

References

- http://docs.openstack.org/kilo/config-reference/content/object-storage-tempurl.html

SSH Console

https://bugs.launchpad.net/ironic/+bug/1526305

This implements console driver ShellinaboxConsole which supports console access for SSH driver (only for virsh virt_type).

Problem description

Currently there is no support to get the console for virtual machines in dev and test environments.

Proposed change

Implements a console driver ShellinaboxConsole that uses ssh+shellinabox to connect to console of virtual machines:

- Use existing ironic/drivers/modules/console_utils module to start/stop shellinabox.
- Add ssh_terminal_port property to CONSOLE_PROPERTIES. This is going to be port on which shellinabox listens locally.
- Add new class ShellinaboxConsole inherited from base.ConsoleInterface in ironic/drivers/modules/ssh.py
- **Implement the following methods in ShellinaboxConsole class.**
  - **`validate()` - Validate the Node console info.**
    - param task : a task from TaskManager.
    - raises : InvalidParameterValue.
    - raises : MissingParameterValue.
  - `start_console()` - Start a remote console for the node.
  - `stop_console()` - Stop the remote console session for the node.

  - `get_console()` - Get the type and connection information about the console.

- Add self variable self.console in PXEAndSSHDriver and AgentAndSSHDriver.

Alternatives

None

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

N/A

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

The following which is already part of driver_info field is required:

- `ssh_address` - IP address or hostname of the node to ssh into.
- `ssh_username` - Username to authenticate as.
- `ssh_virt_type` - Virtualization software to use; must be virsh.

Additionally one field need to be provided with driver_info

- `ssh_terminal_port` - Port to connect to, only required for console access.

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
niu-zglinux

Work Items

Implement `ShellinaboxConsole` class inherited from `base.ManagementInterface`. Implement `validate` , `start_console`, `stop_console`, `get_console`. Add ability to enable pty console in devstack scripts, and leave the log console by default in order not to affect the gate logs.

Dependencies

None

Testing

Unit Testing will be added.

Upgrades and Backwards Compatibility

None

Documentation Impact

None

References

None

5.13 Liberty

5.13.1 4.2

Bare Metal Trust Using Intel TXT

https://bugs.launchpad.net/ironic/+bug/1526280

This uses Intel TXT[4], which builds a chain of trust rooted in special purpose hardware called Trusted Platform Module (TPM)[3] and measures the BIOS, boot loader, Option ROM and the Kernel/Ramdisk, to determine whether a bare metal node deployed by Ironic may be trusted.

Problem description

The bare metal tenant has the ability to introduce rootkits and other malware on the host. Prior to releasing the host to a new tenant, it is prudent to ensure the machine is in a known good state.

Using Intel TXT[4], the TPM[3], Trusted Boot[1], and remote authentication[2], it is possible to confirm that the BIOS, boot loader, Option ROM, and the Kernel/Ramdisk are all in a known good state.

Proposed change

Add a new boot mode, trusted boot:

- Read value capabilities:trusted_boot from flavor. Pass boolean value trusted_boot to ironic.drivers.modules.deploy_utils.switch_pxe_config(). Switch to trusted_boot section.
- Add a new section trusted_boot in PXE Configuration. It will make use of mboot.c32 which supports multiple loading. It loads TBOOT first. TBOOT will measure Kernel/Ramdisk before loading them. PXE config template:

```
label trusted_boot
kernel mboot
append tboot.gz --- {{pxe_options.aki_path}} root={{ ROOT }} ro text
{{ pxe_options.pxe_append_params|default("", true) }} intel_iommu=on
--- {{pxe_options.ari_path}}
```

Alternatives

Secure Boot[5] is used for the same purpose. The main difference is secure boot will verify the signature before executing while trusted boot uses a hardware root of trust and can be configured to verify each component before executing or execute all components and capture measurements (aka extended hash computations) for post verification. So if a node is changed, trusted boot will still boot it up but give a warning to users. Secure boot will not boot it up at all.

They are complementary, both making the cloud more secure. It is recommended to boot nodes with secure boot under uefi and boot nodes with trusted boot under legacy BIOS. The next step is to combine them together but that is out of the scope of this spec.

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

RPC API impact

None

Driver API impact

None

Nova driver impact

Will pass the extra_spec capabilities:trusted_boot=True to Ironic

Ramdisk impact

N/A

Security impact

Increased confidence in bare metal nodes being free of rootkits and other malware. Intel TXT and TPM are leveraged.

Other end user impact

None

Scalability impact

Our experiments indicate handling concurrent attestation requests is linear in the number of requests. Attestation occurs on the node release path, and thus is not latency sensitive.

Performance Impact

There is an extra attestation step during trusted boot which spends several seconds. But for bare metal trust no dynamic attestation requests are entertained. So this is a non-issue.

Other deployer impact

- Create a special flavor with capabilities:trusted_boot=True
- Set `trusted_boot:True` as capability in node.properties.
- **Additionally two items need to be provided with tftpboot/httpboot folder**
  - mboot.c32 - Support multiple loading from /usr/lib/syslinux/mboot.c32
  - tboot.gz - a pre-kernel module to do measurement.
- Set up each machine, enable Intel TXT, VT-x and VT-d and take ownership of the TPM, reboots, and captures the platform configuration register (PCR) values. This is to create the whitelist values that will be registered in the attestation service at initialization time.
- Set up an OAT-Server and create the whitelist with all known types of hardwares from previous step.
- Create customized images with OAT-Client.
- Run a customized script to verify the trust state of nodes when creating instances.

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
tan-lin-good

Work Items

- Add trusted_boot section to pxe_config.template
- Support trusted_boot flag and switch to trusted_boot.
- A dib element to create customized images.

Dependencies

- TBOOT[1]
- OAT[2]
- Hardware Support: TPM and Intel TXT

Testing

Will add unit tests. Planning on adding third party hardware CI testing.

Upgrades and Backwards Compatibility

None. Backwards compatibility is achieved by not requesting trusted bare metal. Custom tenant images are accommodated by deploying an initial standard image that has the OAT client embedded. Today Fedora releases come bundled with the OAT client. This solution approach, while increasing the number of boots preserves us from having to doctor the tenant image by way of injecting the OAT client into the same, or requiring that bare metal users provide images with an OAT client included.

Documentation Impact

Will document usage and benefits. Here is a doc for the technical detail of Bare metal trust: https://wiki.openstack.org/wiki/Bare-metal-trust

References

1. http://sourceforge.net/projects/tboot/
2. https://github.com/OpenAttestation/OpenAttestation
3. http://en.wikipedia.org/wiki/Trusted_Platform_Module
4. http://en.wikipedia.org/wiki/Trusted_Execution_Technology
5. https://review.openstack.org/#/c/135228/
6. http://docs.openstack.org/admin-guide-cloud/compute-security.html#trusted-compute-pools

Cisco IMC PXE Driver

https://blueprints.launchpad.net/ironic/+spec/cisco-imc-pxe-driver

This spec proposes adding a new driver which provides out-of-band non-ipmi based control of UCS C-Series servers through CIMC.

Problem description

Current drivers only allow for control of UCS servers via either IPMI or UCSM, the Cisco UCS C-Series operating in standalone mode can also be controlled via CIMC using its http/s XML API. This provides finer control over the server than IPMI can, and doesnt require the extra infrastructure that UCSM needs.

Proposed change

Power and Management interfaces will be created that understand how to talk to CIMC, and these will be used in conjunction with the PXE boot interface, ISCSI deploy interface and Agent deploy interface to create pxe_cimc and agent_cimc drivers.

The CIMC Power interface will inherit the base PowerInterface and implement:

- get_power_state
- set_power_state
- reboot
- get_properties
- validate

The CIMC Management interface will inherit the base ManagementInterface and implement:

- get_properties
- validate
- get_supported_boot_devices
- get_boot_device
- set_boot_device
- get_sensors_data - This will raise NotImplemented

Alternatives

The alternatives are to use the pxe_ipmi or agent_ipmi driver to control the UCS C-Series via IPMI, or install the infrastructure to manage these servers with UCSM and use the pxe_ucs or agent_ucs driver.

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

When enrolling a node into ironic a user must provide:

- cimc_address - CIMC IP Address
- cimc_username - CIMC Username
- cimc_password - CIMC Password

Additional properties added to ironic.conf in a [cimc] section are:

- max_retry: maximum times to retry any power operation, default: 6
- action_interval: the time to wait in-between power operation retries

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
sambetts

Other contributors:
None

Work Items

- Write CIMC Power Interface and respective unit tests
- Write CIMC Management Interface and respective unit tests
- Create drivers from new and existing interfaces
- Create configuration documentation for pxe_cimc and agent_cimc

Dependencies

This driver requires this installation of the ImcSDK on the node where the ironic conductor will be running.

Testing

Unit tests for the Power and Management interfaces will be provided. Functional testing will be added in the future.

Upgrades and Backwards Compatibility

There should be no compatibility issues introduced by this change.

Documentation Impact

- Writing configuration documentation.
- Updating Ironic documentation section Enabling Drivers: http://docs.openstack.org/developer/ironic/deploy/drivers.html with pxe_cimc and agent_cimc driver related instructions.

References

Cisco Imc Python SDK v0.7.1: https://communities.cisco.com/docs/DOC-37174

Deprecate the bash ramdisk

https://blueprints.launchpad.net/ironic/+spec/deprecate-bash-ramdisk

This spec is a continuation of the blueprint ipa-as-default-ramdisk implemented in the Kilo release. This spec intends to deprecate deployments using the bash script ramdisk.

Problem description

The bash ramdisk is still supported by the drivers prefixed with `pxe_` without any deprecation message. In the Kilo release it was agreed that we should stop supporting the bash ramdisk in the future and we worked on making the IPA ramdisk supported by all drivers in tree.

Also, the bash ramdisk is already lagging behind support for some features, for example cleaning only works with IPA. So now we should start dropping the support for that ramdisk.

Proposed change

We can not simply delete the code that the bash ramdisk uses, therefore we should start adding deprecation messages on the `deploy-ironic` element from diskimage-builder and in the vendor passthru methods `pass_deploy_info` and `pass_bootloader_install_info` which are used by the bash ramdisk to pass the deployment information to Ironic.

Apart from the deprecation messages this spec also proposes freezing the features for the bash ramdisk. No new features should be added to it (like we did to include support for local boot), only bug fixes will be accepted.

Devstack and tempest jobs should also be updated to not use the bash ramdisk anymore.

The element in diskimage-builder and the deprecated code in Ironic should be removed in the Mitaka release cycle of OpenStack.

Alternatives

Continue to support the bash ramdisk for a longer time.

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

Deployer should start replacing the bash ramdisk with the IPA ramdisk. Theres no new configuration needed for it, its a drop-in replacement.

Developer impact

Developers wont be allowed to include any new features to the bash ramdisk, only bug fixes.

Implementation

Assignee(s)

Primary assignee:
lucasagomes <lucasagomes@gmail.com>

Other contributors:
everyone

Work Items

- Update Devstack and tempest to use the IPA ramdisk instead of the bash ramdisk.
- Add deprecation messages on the diskimage-builder `deploy-ironic` element and vendor passthrus `pass_deploy_info` and `pass_bootloader_install_info`.
- Stop accepting new features for the bash ramdisk (code reviews and spec review).
- In the Mitaka release cycle remove the element from *diskimage-builder* and the code that supports the bash ramdisk in Ironic.

Dependencies

None

Testing

Unittests will be added.

Upgrades and Backwards Compatibility

None

Documentation Impact

The documentation should be updated to say that the bash ramdisk is deprecated and the examples should now use IPA instead.

References

Boot Interface in Ironic

https://blueprints.launchpad.net/ironic/+spec/new-boot-interface

This spec talks about refactoring the boot logic out of the current Ironic deploy drivers into a new boot interface.

Problem description

Current we have a DeployInterface in Ironic. All the current implementations of this interface are responsible for two things:

- Booting the bare metal node - both deploy ramdisk and the deployed instance.
- Actual deployment of an image on a bare metal node.

These are two separate functions and therefore should be abstracted separately. This makes it easy to mix and match various boot mechanisms (like PXE, iPXE, virtual media) with various deploy mechanisms (iscsi deploy, agent deploy and various other deploy mechanisms which may be possible like deploying with torrent, multicast, etc in which discussions have been initiated) without duplicating code.

Proposed change

- A new `BootInterface` needs to be added. The interface will recommend the following methods for the implementer:

```
@six.add_metaclass(abc.ABCMeta)
class BootInterface(object):
  """Interface for inspection-related actions."""

  @abc.abstractmethod
  def get_properties(self):
      """Return the properties of the interface.

      :returns: dictionary of <property name>:<property description>
     """

  @abc.abstractmethod
  def validate(self, task):
      """Validate the driver-specific info for booting.

      This method validates the driver-specific info for booting a ramdisk
      and an instance on the node.  If invalid, raises an exception;
      otherwise returns None.

      :param task: a task from TaskManager.
      :raises: InvalidParameterValue
      :raises: MissingParameterValue
      """

  @abc.abstractmethod
  def prepare_ramdisk(self, task, ramdisk_params):
      """Prepares the boot of Ironic ramdisk.

      This method prepares the boot of the deploy ramdisk after reading
      relevant information from the node's database.

      :param task: a task from TaskManager.
      :param ramdisk_params: the options to be passed to the ironic␣
↪ramdisk.
```

(continues on next page)

(continued from previous page)

```
        Different interfaces might want to boot the ramdisk in different
        ways by passing parameters to them.  For example,
        * When DIB ramdisk is booted to deploy a node, it takes the
          parameters iscsi_target_iqn, deployment_id, ironic_api_url,␣
↪etc.
        * When Agent ramdisk is booted to deploy a node, it takes the
          parameters ipa-driver-name, ipa-api-url, root_device, etc.
        Other interfaces can make use of ramdisk_params to pass such
        information.  Different implementations of boot interface will
        have different ways of passing parameters to the ramdisk.
    """

@abc.abstractmethod
def clean_up_ramdisk(self, task):
    """Tears down the boot of Ironic ramdisk.

    This method tears down the boot of the deploy ramdisk after reading
    relevant information from the node's database.

    :param task: a task from TaskManager.
    """

@abc.abstractmethod
def prepare_instance(self, task):
    """Prepares the boot of instance.

    This method prepares the boot of the instance after reading
    relevant information from the node's database.

    :param task: a task from TaskManager.
    """

@abc.abstractmethod
def clean_up_instance(self, task):
    """Tears down the boot of instance.

    This method tears down the boot of the instance after reading
    relevant information from the node's database.

    :param task: a task from TaskManager.
    """
```

- The following new implementations of `BootInterface` will be created.
  - `pxe.PXEBoot` - Booting a bare metal node using PXE
  - `ipxe.IPXEBoot` - Booting a bare metal node using iPXE
  - `ilo.boot.IloVirtualMediaBoot` - Booting a bare metal node using iLO Virtual Media.

> **Note**
>
> Even though IPXEBoot and PXEBoot are in same deploy driver currently, the steps for preparing a bare metal to boot from PXE and iPXE are different (even though they share some common code). We will refactor both of them as separate boot interfaces. The Kilo behaviour of using only either of PXE or iPXE at same time will be retained - drivers will instantiate pxe.PXEBoot or ipxe.IPXEBoot depending on CONF.pxe.ipxe_enabled.

- The code for the above implementations of `BootInterface` will be taken from `pxe.PXEDeploy`, `agent.AgentDeploy`, `ilo.IloVirtualMediaIscsiDeploy` and `ilo.IloVirtualMediaAgentDeploy`. These implementations of `DeployInterface` will be freed of any logic dealing with booting of bare metal node.
- `pxe.PXEDeploy` will be refactored into `pxe.PXEBoot` and `iscsi_deploy.ISCSIDeploy`.
- Each driver will mention what is the `BootInterface` implementation that it wishes to instantiate. For example, the `pxe_ipmitool` driver will look like the following:

```
class PXEAndIPMIToolDriver(base.BaseDriver):
  """PXE + IPMITool driver"""

  def __init__(self):
    self.power = ipmitool.IPMIPower()
    self.console = ipmitool.IPMIShellinaboxConsole()
    self.boot = pxe.PXEBoot()
    self.deploy = iscsi_deploy.ISCSIDeploy()
    self.management = ipmitool.IPMIManagement()
    self.vendor = pxe.VendorPassthru()
    self.inspect = discoverd.DiscoverdInspect.create_if_enabled(
        'PXEAndIPMIToolDriver')
```

> **Note**
>
> It might make sense to rename the drivers to include the boot interface as well as deploy interface after this is implemented. As such, this requires a better thought out process to rename the drivers, address issues of backward compatibility, etc. Hence it is out of scope of this spec. That can be addressed later after this is implemented.

Alternatives

We can continue to keep the boot and deploy logic together but this will lead to code duplications and unnecessary refactorings when additional deploy mechanisms and boot mechanisms are added in the future.

Data model impact

None.

State Machine Impact

None.

REST API impact

None.

RPC API impact

None.

Client (CLI) impact

None.

Driver API impact

This adds the a new `BootInterface` (as described above) which driver writers may use with the deploy drivers. `BootInterface` is not a mandatory interface.

Nova driver impact

None.

Security impact

None.

Other end user impact

None.

Scalability impact

None.

Performance Impact

None.

Other deployer impact

None.

Developer impact

New driver developers adding new deploy mechanisms in Ironic will be encouraged to separate boot and deploy logic so that it can reused easily.

Implementation

Assignee(s)

rameshg87

Work Items

- Add new boot interface
- Create `pxe.PXEBoot`, `ipxe.IPXEBoot` and refactor `pxe.PXEDeploy` into `iscsi_deploy.ISCSIDeploy` to make use of these boot interfaces.
- Refactor `agent.AgentDeploy` to use new `pxe.PXEBoot` and `ipxe.IPXEBoot` (Yes, we are adding iPXE support for agent deploy).
- Create `ilo.boot.IloVirtualMediaBoot`, and refactor `IloVirtualMediaIscsiDriver`, `IloVirtualMediaAgentDriver` to make use of the new boot interface.

Dependencies

None.

Testing

Unit tests will be updated for the new interfaces. Since this change doesnt add any new functionality, the current upstream CI testing should be enough.

Upgrades and Backwards Compatibility

This doesnt break out-of-tree deploy drivers. Still it will be possible to implement deploy drivers for provisioning bare metal nodes without a boot interface- i.e without separate boot and deploy interfaces. This is because the conductor will still be using all the published interfaces of `DeployInterface` for deploying a bare metal node.

This change proposes the addition of new optional boot interface which can be used as a helper for `DeployInterface` and refactors all upstream deploy drivers to follow this logic.

Documentation Impact

Changes to the existing interface will be documented. Also, new developer documentation will be updated to encourage splitting deploy logic into separate boot and deploy interfaces.

References

Not according to this spec, but a POC how it will look like: * https://review.openstack.org/#/c/166512/ * https://review.openstack.org/#/c/166513/ * https://review.openstack.org/#/c/166521/

Make ilo drivers standalone in ironic by removing swift dependency

https://blueprints.launchpad.net/ironic/+spec/remove-swift-dependency-for-ilo-drivers

This spec proposes to remove hard dependency on swift for ilo virtual media drivers. There are standalone use-cases of Ironic (like bifrost) which are capable of deploying nodes without requiring other Openstack services. With this the ilo virtual media drivers can also work standalone similar to other drivers in ironic.

Problem description

Today the ilo drivers (iscsi_ilo and agent_ilo) require swift to host the images like boot_iso (from which the instance boots) and floppy images (used for passing parameters to deploy ramdisk).

Proposed change

The ilo drivers (iscsi_ilo and agent_ilo) can use a web server to host the images required. Swift would be used as a default backend to host boot ISO and floppy images for these drivers.

- The automatic boot_iso created (as required by iscsi_ilo for booting up the instance) and floppy images (as required by both agent_ilo and iscsi_ilo for passing parameters to deploy ramdisk) during deploy process, will be hosted on swift or http web server as per the config variable under *[ilo]* in ironic.conf:

  ```
  use_http_web_server_for_images=True
  ```

  The default value would be *False* and will default to use swift.

- User needs to manually configure the webserver, and add the config options in ironic.conf under *deploy* as:

  ```
  http_server_root = /opt/stack/ironic/data/httpboot
  http_server_url = http://10.10.1.30/httpboot
  ```

  Since the same config variables exists under *[pxe]* and are required by ilo drivers to be able to run standalone, we can deprecate the same in *[pxe]* and move it under *[deploy]*. The above values to the config variables are just an example. They will continue to have the default value as the current config variables `http_url` and `http_root`.

- To add the functionality of take_over() to ilo drivers. This is to enable a case when the conductor node goes down and other conductor takes over the baremetal. The take_over() will be implemented to do regenerate in following scenarios:
  - for wait_call_back state: implement regeneration of floppy images.
  - for active state : implement regeneration of boot ISO.

Alternatives

None.

Data model impact

None.

State Machine Impact

None.

REST API impact

None.

Client (CLI) impact

None.

RPC API impact

None.

Driver API impact

None.

Nova driver impact

None.

Security impact

None.

Other end user impact

None.

Scalability impact

None.

Performance Impact

None.

Other deployer impact

The http web server configuration is out of scope of ironic but it should be configured on every conductor node. User needs to manually configure the web server, and add the config options in ironic.conf under *deploy* as:

```
http_server_root = /opt/stack/ironic/data/httpboot
http_server_url = http://10.10.1.30/httpboot
```

Since the same config variables exists under *[pxe]* and is required by ilo drivers to be able to run standalone, we can deprecate the same in *[pxe]* and move it under *[deploy]*.

Developer impact

None.

Implementation

Assignee(s)

Primary assignee:
agarwalnisha1980

Work Items

- To modify ipxe to use config variables from *[deploy]* section.
- To enable support for using webserver in ilo drivers.
- To implement take_over() for ilo_drivers for `use_web_server=True`

Dependencies

None.

Testing

- Mocked unit tests would be added.
- Functional tests would be done to ensure that the deploy works fine for ilo drivers with swift as backend or http webserver as backend.

Upgrades and Backwards Compatibility

The ilo drivers will continue to work with swift as backend. The iPXE code will continue to work for config options in either of the section.

Documentation Impact

It is required to be documented.

References

None.

5.13.2 4.0

Cisco UCS PXE driver

https://blueprints.launchpad.net/ironic/+spec/cisco-ucs-pxe-driver

This blueprint proposes adding new driver that supports deployment of Cisco UCS Manager (UCSM) managed B/C/M-series servers.

In this blueprint servers, nodes are used interchangeably that denotes Cisco UCS B/C/M-series servers.

Problem description

Current Ironic drivers require IPMI protocol to be enabled on all Cisco UCS B/C/M-series servers in order to manage power operations. For security reasons from UCS Manager version 2.2.2 IPMI protocol is disabled by default on all servers.

Instead of using IPMI protocol, this blueprint proposes new driver to manage Cisco UCS B/C/M-series servers using Cisco UCS PySDK.

Proposed change

New power and management interfaces will be added as part of pxe_ucs driver. This driver uses the Cisco UCS PySDK for communicating with UCSM.

This driver uses

- pxe.PXEDeploy for PXE deployment operations
- ucs.power.Power for power operations
- ucs.management.UcsManagement for management interface operations

UCS Manager provides python SDK with which user can perform various operations, like controlling the power of nodes, enabling the ports, associating the servers etc. Physical and Logical entities in UCSM are represented as ManagedObject.

- Power management:

  Controlling the power is similar to modifying the property of the corresponding ManagedObject. LsPower is the ManagedObject that represents the Power of service-profile. As part of managing the power, this provider modifies state property of LsPower ManagedObject. UCSM takes care of the rest.

- Management Interface:

  This interface allows the user to get and set the boot-order on UCS B/C/M servers. LsbootDef is the ManagedObject that represents the boot-order of service-profile. This interface reads and updates LsbootDef ManagedObject appropriately for get and set boot-device operations. get_sensor_data() implementation is not in scope of this spec. A separate spec will be submitted.

Alternatives

The IPMI Pxe driver could be used with Cisco UCS B/C/M-series servers, if IPMI protocol is explicitly enabled overriding the default settings in UCS Manager.

Data model impact

None

RPC API impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

Driver API impact

None

Nova driver impact

None

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

The following driver_info fields are required while enrolling node into ironic:

- ucs_address: UCS Manager hostname/ip-address
- ucs_username: User account with admin/server-profile access privilege
- ucs_password: User account password
- ucs_service_profile: service_profile DN (DistinguishedName) being used for this node.

The following parameters are added in to the newly created [ucs] section in the ironic configuration file which is typically located at /etc/ironic/ironic.conf.

- max_retry: maximum number of retries, default value is set to 5.
- action_timeout: seconds to wait for power action to be completed default value is 30 seconds, there is no explicit maximum limit.

Developer impact

None

Implementation

Assignee(s)

Primary assignee: saripurigopi

Other contributors: vinbs

Work Items

- Add new pxe_ucs driver, extending power and management interface APIs.
- Writing and unit-test cases for pxe_ucs driver.
- Writing configuration documents.

Dependencies

This driver requires Cisco UCS Python SDK installed on the conductor node.

Testing

Unit-tests will be implemented for new pxe_ucs driver. tempest test suite will be updated to cover the pxe_ucs driver. Continuous integration (CI) support will be added for Cisco UCS B/C/M series servers.

Upgrades and Backwards Compatibility

This driver will not break any compatibility with either on REST API or RPC APIs.

Documentation Impact

- Writing configuration documents.
- Updating Ironic documentation section Enabling Drivers: http://docs.openstack.org/developer/ironic/deploy/drivers.html with pxe_ucs driver related instructions.

References

Cisco UCS PySdk:https://github.com/CiscoUcs/UcsPythonSDK

DRAC vendor passthru for BIOS settings management

https://blueprints.launchpad.net/ironic/+spec/drac-bios-mgmt

This spec will expose the vendor passthru methods needed to let external services get, set, and commit changes to the BIOS settings. This spec will assume that the external service knows exactly what it is doing with the specific hardware it will manage, and it will not attempt to standardize, normalize, or simplify the exposed settings in any way beyond some minimal convenience measures and what is needed to map between XML and JSON.

Problem description

Tuning a server for a particular workload and power consumption profile involves many different parameters to be tweaked, several of which must be tuned at the level of the system firmware. On Dell systems, doing that out-of-band involves talking to the DRAC to have it make changes on your behalf.

Proposed change

To expose these changes, I propose to add 4 vendor passthrough methods to the drac driver:

- `get_bios_config` - Gather the current BIOS configuration of the system from the drac and return it. This will return a JSON structure that contains the current BIOS configuration of the system. It has no parameters.
- `set_bios_config` - Queue up changes to the BIOS configuration for later committal. This will accept the same JSON data structure that `get_bios_config` returns. It will raise an exception if any of the changed settings does not validate as a proper change, if you try to change a readonly setting, or if the changes cannot be queued up in the DRAC for some other reason.
- `commit_bios_config` - Commit a set of queued up BIOS configuration changes, and schedules a system reboot if is needed to commit the changes.
- `abandon_bios_config` - Abandon a set of queued up BIOS configuration changes.

For this spec, only changes that can be made though the DCIM Bios and Boot Management Profile will be considered, although there are many other BIOS and hardware parameters that can be changed by the DRAC.

Alternatives

Create a common BIOS interface which most vendors will agree to. This is a longer term solution which should replace or drive this implementation in the future.

Data model impact

None.

REST API impact

Four new API calls:

- `get_bios_config` - a GET method. Takes no args, returns a blob of JSON that encapsulates the BIOS configuration parameters of the system.
- `set_bios_config` - a POST method. Takes a blob of JSON that has the same format and settings returned by get_bios_config. Returns the following status codes:
  - 200 if the set of proposed new parameters passed validation and was accepted for further processing.
  - 204 if the set of proposed new parameters passed validation, but did not actually change the BIOS configuration.
  - 409 if the set of proposed new parameters contains a parameter that cannot be set to the requested value, either because the parameter is read-only or the proposed new parameter is not valid.

  - 403 if there are already proposed changes in the process of being committed.

- `commit_bios_config` - a POST method. Takes no args, commits the changes made by `set_bios_config` calls and schedules a system reboot to apply the changes if needed. Returns the following status codes:
  - 202 if a commit job was created, and the system requires a reboot.
  - 200 if the settings were committed without needing a reboot.
  - 204 if there were no settings that needed to be committed.
  - 403 if there are already proposed changes in the process of being committed.
- `abandon_bios_config` - a POST method. Takes no arguments, and abandons any changes made by `set_bios_config` calls that have not been committed by a `commit_bios_config` job. Returns the following status codes
  - 200 if the proposed changes were successfully dequeued.
  - 204 if there were no proposed changes to dequeue
  - 403 if the proposed changes are already in the process of being committed.

RPC API impact

None

Driver API impact

None

Nova driver impact

None for now. If this spec gets generalized to other drivers, we will need to figure out what (if any) capabilities should be exposed to Nova.

Security impact

It is quite possible to render a system unbootable through this API, as it allows you to enable and disable a wide variety of hardware and mess with the boot order indiscriminately.

Other end user impact

None

Scalability impact

None

Performance Impact

None.

Other deployer impact

None

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
victor-lowther

Work Items

- Create and implement DracVendorPassthruBios class

Dependencies

- This feature depends on the python bindings of the OpenWSMAN library which we already use for the rest of the DRAC driver.
- This feature requires 11th or higher generations of Dell PowerEdge servers.

Testing

- Unit tests
- 3rd-party CI: I will try to implement it in parallel with implementing this driver, provided I can source sufficient internal resources and appropriate network connectivity.

Upgrades and Backwards Compatibility

None expected, and there should be no stability guarantee for this API.

Documentation Impact

User documentation should mention this vendor passthrough API.

References

DCIM Bios and Boot Management Profile

Add enroll state to the state machine

https://blueprints.launchpad.net/ironic/+spec/enroll-node-state

This blueprint introduces a new state called `enroll`, which we previously agreed to introduce in the new state machine spec.

Problem description

Currently nodes on creation are put into `available` state, which is designed as a replacement for `NOSTATE`. Such nodes will instantly be available to Nova for deployment, provided they have all the required properties.

However, the new state machine lets the operator perform inspection on a node and zapping of a node. The state machine allows for them to be done before a node reaches the `available` state.

Even worse, the cleaning feature introduced in Kilo cycle should also happen before a node becomes `available`.

Proposed change

- Add a new state `enroll`, from which a node can transition into the `manageable` state by an action called `manage`.
- `manage` transition will cause power and management interfaces to be validated on the node. Also an attempt will be made to get the power state on a node to actually verify the supplied power credentials.

  On success, the node will go to the `manageable` state. On failure, it will go back to the `enroll` state and `last_error` will be set.
- Disable the sync_power_state for nodes in the `enroll` state, as nodes in this state are not expected to have valid power credentials.
- Introduce a new API microversion, making newly created nodes appear in the `enroll` state instead of the `available` state.

  After that the client-server interaction will be the following:

  - new client (with new API version) + new server: nodes appear in the `enroll` state.
  - new client + old server: client gets a response from the server stating that node is in `available` (or `none`) state. Client issues a warning to the user.
  - old client (or new client with old API version) + new server: due to versioning the node will appear in `available` (or `none`) state.
- Document that we are going to make a breaking move to the `enroll` state by default.
- Update DevStack gate to explicitly request the new microversion and fix the tests.
- Release a new version of `ironicclient` defaulting to this new microversion. Clearly document this breaking change in upgrade notes.

Alternatives

We can ask people to manually move nodes to the `manageable` state before inspection or zapping. We also wont validate power and management interfaces.

Data model impact

None

State Machine Impact

- `enroll` becomes a valid node state with transitions to:
  - `verifying` via `manage` action
- `verifying` becomes a valid transient node state with transitions to:
  - `manageable` on `done`
  - `enroll` on `fail` and `last_error` will be set.

REST API impact

- Add new API microversion. When it is declared by client, node creation API should default to creating nodes in the `enroll` state.

Client (CLI) impact

- New release of client will be issued defaulting to the new microversion (and breaking many flows).

RPC API impact

None

Driver API impact

None

Nova driver impact

- Double check that Nova driver wont use nodes in `enroll` state
- Sync `nova/virt/ironic/ironic_states.py` for the sake of consistency

No functionality impact expected.

Security impact

None expected

Other end user impact

With the new microversion, nodes will appear in the `enroll` state. Two more steps should be taken to make them available for deploy: `manage` and `provide`. Cleaning will happen, if enabled.

Scalability impact

None

Performance Impact

None

Other deployer impact

See *Upgrades and Backwards Compatibility*.

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
Dmitry Tantsur, IRC: dtantsur, LP: divius

Other contributors:
None

Work Items

- Create new states and transitions
- Introduce new microversion with node defaulting to `enroll` on creation
- Make sure our tests do not break (fix devstack etc)
- Default ironicclient to the new microversion

Dependencies

None

Testing

- Tempest tests should be modified to test `enroll` state.

Upgrades and Backwards Compatibility

- Change is backwards compatible, while its not the default in ironicclient.
- Once new microversion is the default in ironicclient, it will break existing flows, when explicit microversion is not in use.

Documentation Impact

- Working with the new state and the transition should be documented
- Upgrade notes should be updated

References

Move Ironic to a feature-based release model

https://blueprints.launchpad.net/ironic/+spec/feature-based-releases

At the Liberty design summit in Vancouver, the Ironic team generally agreed to move the Ironic project to a feature-based release model.[0] This is an informational spec to lay out why we are doing this and how it will work.

Problem description

Ironic currently uses the traditional OpenStack release model.[1] This involves:

- One release every six months.
- A feature freeze prior to cutting this release, typically for a duration of six weeks. Only bug fix work and docs work should be done in this period.
- An optional spec freeze a few weeks prior to the feature freeze.
- After the feature freeze period, a stable branch is forked from master and a release candidate period begins. Critical bugs fixed on master during this period may be backported to the stable branch for the final release.
- After about four weeks of iterating on release candidates, the final version is released. This stable release may receive critical bug fixes for a longer period of 9-15 months.

This model creates a few problems. The primary problem created is that development happens in peaks and valleys. Developers and vendors realize that the feature freeze is coming up, and attempt to merge features quickly before the freeze. This creates a flurry of both patches and reviews, which has the tendency to burn out core reviewers. Then feature freeze begins. Features cannot be landed and development slows, almost to a halt. This typically takes six weeks. Once the next cycle opens for feature development, code starts to trickle through but most folks are working on summit planning and taking a break from the stress of the last release. This essentially causes *10 weeks* of developer downtime *per cycle*, or 20/52 weeks per year.

Proposed change

We should release Ironic (roughly) following the independent release model defined by the OpenStack governance.[2]

Note

As of July 15, 2015 [10], ironic follows the cycle-with-intermediary release model [11]. This model, which better matches this specification, was created by the OpenStack governance after this specification had been approved.

There are a few things of note here.

- We should release Ironic when major features or bug fixes are complete on the master branch. This will be at the discretion of Ironics PTL or a designated release manager.
- We should continue to release a final release every six months, to continue to be part of the coordinated release.[3]
- In lieu of a feature freeze, Ironics stable release should come from a normal Ironic release that happens around the same time that other integrated projects begin the release candidate period. For example, for Liberty[4], Ironic should aim to make a release in the second half of September. This will become equivalent to the stable/liberty branch. Release candidates should be built from

this branch and may receive bug fixes. Stable releases should be eligible for receiving backports following the current process.

- Ironic releases should roughly follow SemVer[5] the difference being that the major version should be bumped for significant changes in Ironics functionality or code. Ironic should never do a release without an upgrade path from a prior release, so in traditional SemVer the major version would never change. We should bump minor and patch versions as needed for minor feature and bug fix releases.
- Ironic releases should be published to PyPI.
- Specs should be no longer be targeted to a particular release folks should just work on specs and code continuously. Features will be released as they land. The ironic-specs repo should change to:
  - One approved directory where all specs live initially.
  - Separate $version-implemented directories that house the specs that were implemented in $version of Ironic. Specs are moved to this directory when the work is completed.
  - Create a placeholder in the old location which indicates which release of Ironic the work was completed in, with a link to the new location. This will keep older links from breaking.
- Leading up to all releases, reviewers should honor a soft freeze period of a few days to a week. Code that is risky in terms of breakage should probably not land at this time; quick bug fixes, driver changes, and other less risky changes should be okay to land in most cases. The PTL or designated release manager must be sure to communicate upcoming releases and these freezes well.
- Ironic may need to decouple from global requirements during Dep Freeze[7]. During OpenStacks feature freeze, the master branch of global-requirements is locked. It should be okay to accept changes to Ironics requirements.txt on the master branch that are blocked in global-requirements by the Dep Freeze, as the stable branch has already been cut by this point. This should only happen on an as-needed basis as the problem arises; not by default. We also may need to temporarily drop the gate-ironic-requirements job during this time. Finally, any patches to Ironic changing requirements.txt should also have a patch to global-requirements with a general looks good from a global-requirements core reviewer.
- Folks have discussed using feature branches for larger chunks of work. While these can be useful, we must be sure to only use them when absolutely necessary as feature branches can carry a large amount of pain. We should prefer feature flags[9] over feature branches where possible.

Alternatives

Continue on the status quo. :(

Data model impact

None.

State Machine Impact

None.

REST API impact

None.

Client (CLI) impact

None. The client will continue to release independently, and likely more often than the server.

RPC API impact

None.

Driver API impact

None.

Nova driver impact

As the Nova driver is released with Nova, it will not change in terms of the way it is released.

Security impact

As only stable releases will receive backports, security bugs in other releases should be fixed and released ASAP. An advisory should be published that encourages users to upgrade to the new release.

Stable branches should continue to receive backports for security bug fixes.

Intermediate releases will not receive backports for security patches. Any security bug in an intermediate release should be fixed and released with the appropriate version bump. Whether the version change is major/minor/patch may depend on what else has landed on master and will be released with the patch.

Other end user impact

End users will get features shipped to them more quickly.

Scalability impact

None.

Performance Impact

None.

Other deployer impact

Deployers will receive changes and features more quickly. Those that do not wish to do so may continue to consume the six-month integrated release.

Developer impact

All the productivity.

Implementation

Assignee(s)

PTL, designated release manager (if one exists), and core reviewers.

Work Items

- Switch to semver.
- Start releasing independently.
- Document the process in the developer docs.[8]

Dependencies

None.

Testing

None.

Upgrades and Backwards Compatibility

This should cause upgrades to be smaller and thus less impactful.

Documentation Impact

We should write a page in our developer docs about the process, including the changes to the specs process.

References

[0] https://etherpad.openstack.org/p/liberty-ironic-scaling-the-dev-team

[1] https://governance.openstack.org/tc/reference/tags/release_at-6mo-cycle-end.html

[2] https://governance.openstack.org/tc/reference/tags/release_independent.html

[3] https://governance.openstack.org/tc/reference/tags/integrated-release.html

[4] https://wiki.openstack.org/wiki/Liberty_Release_Schedule

[5] https://semver.org/

[6] http://lists.openstack.org/pipermail/openstack-dev/2015-May/065211.html

[7] https://wiki.openstack.org/wiki/DepFreeze

[8] https://docs.openstack.org/developer/ironic/

[9] https://en.wikipedia.org/wiki/Feature_toggle

[10] https://review.openstack.org/#/c/202208/

[11] http://governance.openstack.org/reference/tags/release_cycle-with-intermediary.html

iRMC Virtual Media Deploy Driver for Ironic

https://blueprints.launchpad.net/ironic/+spec/irmc-virtualmedia-deploy-driver

The proposal presents the work required to add support for deployment features for FUJITSU PRIMERGY iRMC, integrated Remote Management Controller, Drivers in Ironic.

Problem description

FUJITSU PRIMERGY servers are capable of booting from virtual media, but Ironic lacks a driver which can utilize this for PXE/TFTP less deployment.

Proposed change

Adding new iRMC Drivers, namely iscsi_irmc and agent_irmc, to enable PXE/TFTP less deployment capability to provision PRIMERGY bare metal nodes (having iRMC S4 and beyond) by booting the bare metal node with virtual media using NFS or CIFS from a conductor node to deploy an image.

iRMC virtual media deploy driver is basically same as iLOs. However comparing iRMC deploy driver with iLO deploy driver, the only significant change is the location of virtual media images, specifically deploy ISO image, floppy FAT image and boot ISO image. The location where iRMC deploy driver places the created floppy FAT image and the boot ISO image is on an NFS or CIFS server, while the location where iLO deploy driver creates is on Swift Object Storage Service. The location where iRMC mounts the three images is from an NFS or CIFS server, while the location where iLO mounts is from the http temp-url generated for the Swift object Service. The other parts are common.

Before starting Ironic conductor, however, deployer has to set up operating system properly so that Ironic conductor mounts the NFS or CIFS shared file system on path /remote_image_share_root as default, the path is configurable in the ironic configuration file. This driver checks this mount at start up time.

The NFS or CIFS server can be located anywhere in the network as long as iRMC and Ironic conductor can reach it. The NFS or CIFS server and the network path should be redundant so that a bare metal node wont fail to boot. For this reason, Ironic conductor is not recommended to be used as the NFS or CIFS server for high available environments.

The iRMC deploy module uses python-scciclient package to communicate with ServerView Common Command Interface (SCCI) via HTTP/HTTPS POST protocol.

The details of ServerView Common Command Interface (SCCI) is described in FUJITSU Software ServerView Suite, Remote Management, iRMC S4 - integrated Remote Management Controller

Alternatives

Other drivers such as the following can be used, but there are no drivers that can boot using virtual media.

- iRMC driver for PXE (pxe_irmc) which can boot in either Legacy mode or UEFI. The details of the boot mode control is described in iRMC Management Driver for Ironic.
- IPMI driver for PXE (pxe_ipmitool) which can boot only in Legacy mode.

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Security impact

- Admin credentials will be stored unencrypted in the Ironic configuration file, and the DB which will be visible in the driver_info field of the node when a node-show is issued. But only the ironic admin user will have access to the Ironic configuration file and the DB.
- NFS and CIFS have very similar functionality, but have different security model. NFS server authenticates NFS client based on NFS client host IP address, while CIFS server authenticates CIFS client based on user identity. For this reason, this driver deals with CIFS as default, and NFS as alternative. Advanced deployer could use NFSv4 and Kerberos to authenticate NFS client based on user identity. Detail information is available in each linux distribution manual ([1],[2])

Other end user impact

None

Scalability impact

NFS or CIFS server would become performance bottleneck in case of managing a large number of bare metal nodes such as more than 1,000 nodes. In that case, the load needs to be distributed and balanced among multiple of NFS or CIFS server settings ([3],[4],[5])

[1] https://help.ubuntu.com/community/NFSv4Howto
[2] http://docs.fedoraproject.org/en-US/Fedora/17/html/FreeIPA_Guide/kerb-nfs.html
[3] https://help.ubuntu.com/community/HighlyAvailableNFS
[4] https://wiki.samba.org/index.php/Samba_CTDB_GPFS_Cluster_HowTo
[5] https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Cluster_Administration/ch-clustered-samba-CA.html

Performance Impact

None

Other deployer impact

- The following parameters are required in the [irmc] section of the ironic configuration file which is typically located at /etc/ironic/ironic.conf in addition to the parameters defined in iRMC Power Driver for Ironic
  - remote_image_share_root: Ironic compute nodes NFS or CIFS root path (string value). The default value is /remote_image_share_root.
  - remote_image_server: IP of remote image server
  - remote_image_share_type: The share type (NFS or CIFS) of virtual media. The default value is CIFS.
  - remote_image_share_name: The share name of remote_image_server. The default value is share.
  - remote_image_user_name: user name of remote_image_server
  - remote_image_user_password: password of remote_image_user_name
  - remote_image_user_domain: domain name of remote_image_user_name. The default value is .
- The following driver_info field is required to support iRMC virtual media in addition to the fields defined in iRMC Power Driver for Ironic.
  - irmc_deploy_iso: deploy ISO image which is either a file name relative to remote_image_share_root, Glance UUID, Glance URL or Image Service URL.
- The following instance_info field is optional.
  - irmc_boot_iso: boot ISO image file name relative to remote_image_share_root, Glance UUID, Glance URL or Image Service URL. If it is not specified, the boot ISO is created automatically from registered images in Glance.
- In order to use iRMC virtual media deploy driver, iRMC S4 and beyond with iRMC a valid license is required. Deployer is notified by error message if the iRMC version and/or the license is not valid.
- In order to deploy and boot ISO image via virtual media, an NFS or CIFS server is required. The NFS or CIFS server has to be reachable from both iRMC and Ironic conductor.

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
Naohiro Tamura (naohirot)

Other contributors:
None

Work Items

- Add iRMC Drivers (iscsi_irmc, agent_irmc)
- Implement iRMC virtual media deploy module for the iRMC Drivers by reusing and refactoring some part of the code from the current iLO deploy driver.

Dependencies

- This feature requires iRMC S4 and beyond that is at least BX S4 or RX S8 generation of FUJITSU PRIMERGY servers.
- This feature uses python-scciclient package.
- This feature depends on iRMC Power Driver for Ironic and iRMC Management Driver for Ironic.

Testing

- Unit Tests
- Fujitsu plans Third-party CI Tests

Upgrades and Backwards Compatibility

None

Documentation Impact

The required driver_info fields and [irmc] section parameters in the ironic configuration file need be included in the documentation to instruct operators how to use Ironic with iRMC.

References

- FUJITSU Software ServerView Suite, Remote Management, iRMC S4 - integrated Remote Management Controller
- iRMC Power Driver for Ironic
- iRMC Management Driver for Ironic
- python-scciclient package
- iLO Virtual Media iSCSI Deploy Driver
- iLO IPA Deploy Driver
- Automate UEFI-BIOS Iso Creation
- Support for non-glance image references

Open CloudServer (OCS) power driver

https://blueprints.launchpad.net/ironic/+spec/msft-ocs-power-driver

This blueprint adds support for the Open CloudServer (OCS) v2.0 power interface in Ironic. The OCS design and specs have been contributed by Microsoft to the Open Compute project.

Problem description

The OCS chassis system includes a chassis manager server which exposes a REST API to manage the individual blades, replacing traditional protocols like IPMI. The REST API service itself is open source (Apache 2 license).

In order to be able to execute power and management actions on OCS blades, the corresponding interfaces need to be implemented.

Furthermore, the OCS REST API also supports a serial console interface for individual blades that can be supported in Ironic.

Proposed change

The proposed implementation consists of a driver implementation. A client will be provided to abstract the OCS service REST API calls, which in turn can be referenced by the power, management and console interfaces.

Both UEFI and legacy BIOS boot modes are supported and can be specified by the user as part of the properties/capabilities.

Driver properties that can be specified by the user:

msftocs_base_url

Base url of the OCS chassis manager REST API, e.g.: http://10.0.0.1:8000. Required.

msftocs_blade_id

Blade id, must be a number between 1 and the maximum number of blades available in the chassis. In the current configuration OCS chassis have a maximum of 24 or 48 blades. Required.

msftocs_username

Username to access the chassis manager REST API. Required.

msftocs_password

Password to access the chassis manager REST API. Required.

Alternatives

No alternatives are available for the OCS case.

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Security impact

The interaction between Ironic and the OCS chassis manager involves REST API calls, using HTTP basic authentication and potentially NTLM authentication in the future.

The HTTP credentials are provided by the user as part of the driver properties and need to be passed to the REST API service. It is highly recommended to employ HTTPS for transport encryption in any production environment.

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

None

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
<alexpilotti>

Other contributors:
<atuvenie>

Work Items

- Power and management interfaces
- Console interface

Dependencies

None

Testing

Potential continuous integration system integrated with Gerrit / Zuul. The challenge is that a non trivial amount of OCS resources is required for this purpose.

Upgrades and Backwards Compatibility

None

Documentation Impact

The driver should be documented in a way similar to other Ironic drivers under http://docs.openstack.org/developer/ironic/index.html

References

- OCS design and specs: http://www.opencompute.org/wiki/Server/SpecsAndDesigns
- Chassis Manager sources: https://github.com/MSOpenTech/ChassisManager

Determinable supported boot device list

https://blueprints.launchpad.net/ironic/+spec/supported-boot-device-list

This blueprint proposes to add the facility to return a determinable list of supported boot devices that is specific to the node being queried.

Problem description

Current driver interface get_supported_boot_devices() does not provide for a means to access the node being queried, as a result a determinable list of supported boot devices that are specific to this node cannot be calculated.

Example usage, if a node uses an architecture other than x86 e.g. SPARC, then PXE boot is not supported, and therefore should not be contained in the returned list. SPARC architecture supports WANBOOT and this should be returned. However to determine this the nodes property cpu_arch needs to be accessed.

Proposed change

Add task parameter to get_supported_boot_devices() interface.

By providing a task parameter to the get_supported_boot_devices() interface a driver can then access the node specific to this query e.g. task.node, and from there can look at a nodes properties and correctly determine the list of supported boot devices to return.

This change would be backwards compatible, using inspect to determine if a specific driver as implemented the task parameter or not, and showing a deprecation warning if task parameter has not been implemented.

This deprecation of get_suported_boot_devices() without task parameter will be done in the next release, after which get_supported_boot_devices() without the task parameter will not be supported.

Alternatives

The only method to access the specific node for which a get_supported_boot_devices request is being made is via passing in the task argument.

Alternative 1: Restrict all drivers to only be able to register nodes of a specific architecture. Then the underlying driver API for get_supported_boot_devices() would just assume that this node must be of a specific architecture and always return a static list for that architecture.

This approach is limiting and would result in an unnecessary increase in the number of drivers available for ironic, and a unnecessary potential duplication of code.

Alternative 2: Utilize set_boot_device(task, device), this method in most cases will validate if a device is supported for this node. A task parameter is being passed in here, and thus the node can be accessed to determine if device is supported for this specific node. The issue here is that it would have to be called a number of times to determine a list of supported devices, and each successful call would result in an IPMI call to actually set the boot device for the node which would be unacceptable and inefficient.

This approach is not complete and does not solve the scenario where get_supported_boot_devices() is called via the ironic CLI. This would still result in a potentially incorrect list of devices being returned to the user.

Alternative 3: This is more of an addendum to the proposed solution. Add a new essential property for a node called supported_boot_devices. This property would be populated during node inspection, and could then be queried to determine the list of supported boot devices.

Access to the node however would still be required, so passing of the task parameter to get_supported_boot_devices would still be required.

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

RPC API impact

None

Driver API impact

get_supported_device_list() is a member of the standard Management interface.

This change would effect all drivers but would be implemented so that its backward compatible, using inspect to determine if a driver supports the task argument or not. If not a deprecation warning would be shown.

Nova driver impact

None

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

None

Developer impact

All in-tree driver and unit tests will be amended to include the new task parameter. Out of tree drivers will see a deprecation warning until they implement the new task parameter.

Implementation

Assignee(s)

Primary assignee:
mattkeenan

Work Items

- Add task to get_supported_devices_list() in base driver
- Add new management property definition to check if driver supports task parameter or not.
- Update all in tree drivers adding task parameter
- Update all unit tests affected by change

Dependencies

None

Testing

Will update all affected unit tests

Upgrades and Backwards Compatibility

Backwards compatibility is achieved by using pythons inspect module to determine if a drivers get_supported_device_list() implementation includes a task parameter or not. If not it will be called without the task parameter and a deprecation warning will be shown.

Documentation Impact

None

References

Bug: https://bugs.launchpad.net/ironic/+bug/1391598 Review: https://review.openstack.org/#/c/188466

UEFI Secure Boot support for pxe_iLO driver

Include the URL of your launchpad blueprint:

https://blueprints.launchpad.net/ironic/+spec/uefi-secure-boot-pxe-ilo

As part of Kilo release UEFI secure boot support was enabled for all the iLO drivers except pxe_ilo. It is important to have this feature supported for pxe_ilo driver so that security sensitive users of pxe_ilo driver could deploy more securely using Secure Boot feature of the UEFI. This spec proposes UEFI Secure Boot support in baremetal provisioning for pxe_ilo driver.

Problem description

Secure Boot is part of the UEFI specification (http://www.uefi.org). It helps to make sure that node boots using only software that is trusted by Admin/End user.

Secure Boot is different from TPM (Trusted Platform Module). TPM is a standard for a secure cryptoprocessor, which is dedicated microprocessor designed to secure hardware by integrating cryptographic keys into devices. Secure Boot is part of UEFI specification, which can secure the boot process by preventing the loading of drivers or OS loaders that are not signed with an acceptable digital signature.

When the node starts with secure boot enabled, system firmware checks the signature of each piece of boot software, including firmware drivers (Option ROMs), boot loaders and the operating system. If the signatures are good, the node boots, and the firmware gives control to the operating system.

The Admin and End users having security sensitivity with respect to baremetal provisioning owing to the workloads they intend to run on the provisioned nodes would be interested in using secure boot provided by UEFI.

Once secure boot is enabled for a node, it cannot boot using unsigned boot images. Hence it is important to use signed bootloaders and kernel if node were to be booted using secure boot.

This feature has been enabled for iscsi_ilo and agent_ilo driver during Kilo release. It needs to be enabled for pxe_ilo driver. This needs pxe_ilo driver should support signed UEFI bootloader for the nodes to boot in the UEFI secure boot environment.

Proposed change

This spec proposes to support UEFI secure boot for pxe_ilo driver and grub2 as an alternate bootloader for UEFI deploy for PXE drivers.

Preparing the environment

- The operator informs the Ironic using the `capabilities` property of the node. The operator may add a new capability `secure_boot=true` in `capabilities` within `properties` of that node. This is an optional property that can be used if node needs to be provisioned for secure boot. By default the behavior would be as if this property is set to false. The inspection feature in iLO drivers can auto discover secure boot capability of the node and create node capability into that node object.
- If the user has `secure_boot` capability set in the flavor, pxe_ilo has ability to change the boot mode to UEFI and prepare the node for the secure boot on the fly using proliantutils library calls.
- Even if the `secure_boot` capability is set to `true` in the nodes `properties/capabilities`, node can be used for normal non-secure boot deployments. Driver would use the `secure_boot` capability information from the nodes `instance_info` field to provision node for UEFI secure boot.

Preparing flavor for secure boot

- The `extra_specs` field in the nova flavor should be used to indicate secure boot. User will need to create a flavor by adding capabilities:secure_boot=true to it.
- iLO driver will not do secure boot if secure_boot capability flavor is not present or set to False. Nova scheduler will use secure_boot capability as one of the node selection criteria if secure_boot is present in extra_spec. If secure_boot is not present in extra_spec then Nova scheduler will not consider secure_boot capability as a node selection criteria.

- Ironic virt Driver will pass the flavor capability information to the driver as part of instance_info. Having capability information as part of instance_info would help driver in preparing and decommissioning the node appropriately. With respect to secure boot feature, this information would be used by pxe_ilo driver for:-
  - During provisioning, driver can turn on the secure boot capability to validate signatures of bootloaders and kernel.
  - During teardown, secure boot mode would be disabled on the node.

Preparing bootloader and deploy images

To support UEFI secure boot for pxe_ilo driver, pxe driver for Ironic should support signed UEFI bootloader. Currently elilo is the default UEFI bootloader for all pxe drivers. Not all major linux distros ship signed elilo bootloader. They ship signed grub2 bootloader.

Enabling grub2 bootloader requires steps similar to elilo. Steps are:-

- Copy signed shim and grub2 bootloader files into tftproot directory as bootx64.efi and grubx64.efi respectively .
- Create a master grub.cfg file under /tftpboot/grub
- Contents of master grub.cfg would look something like this. set default=master set timeout=5 set hidden_timeout_quiet=false

  menuentry master { configfile /tftpboot/$net_default_ip.conf }

  This master grub.cfg gets loaded first during PXE boot. It tells grub to refer to the node specific config file in tftproot directory configured for PXE. The name of config file is coined using DHCP IP address that would be allocated to the node. This is to ensure that multiple grub.cfg files could be created for parallel deploys. The contents of $net_default_ip.conf is dynamically filled by PXE driver using grub template file.

Ironic needs to support grub2 as an alternate UEFI bootloader for following reasons:-

- No active development happening on elilo
- All major linux distributions are supporting grub2 as a default UEFI bootloader.
- All major linux distributions provide signed grub2 bootloader which could be used in UEFI secure boot deploy with distro supplied cloud images. Otherwise users would need to build their own signed images for secure boot deploy.
- signed grub2 can be used for normal UEFI deploys as well.

All major linux distros ship their self signed grub2 and also provide Microsoft UEFI CA signed shim bootloader. The shim bootloader contains the UEFI signature of respective distros.

When node boots up using pxe, it loads Microsoft signed shim boot loader which in turn loads the distro signed grub2. Distro signed grub2 can validate and load the distro kernel. Shim bootloader is required as it is signed using Microsoft UEFI CA signature and recognizes corresponding linux vendors certificate as a valid certificate. Secure boot enabled HP Proliant UEFI systems are pre-loaded with Microsoft UEFI CA signatures. User signed images can be supported but user need to manually configure their keys to HP Proliant system ROM database using Proliant tools.

User can configure grub2 as a bootloader by changing the following existing variables in /etc/ironic/ironic.conf under pxe section: uefi_pxe_config_template uefi_pxe_bootfile_name

Alternatives

Add support for signed grub2 as a default UEFI bootloader in Ironic. But such a change would have backward compatibility impact.

Data model impact

None

State Machine Impact

None

REST API impact

None

RPC API impact

None

Client (CLI) impact

None

Driver API impact

None

Nova driver impact

None

Security impact

This enhances security. Only correctly signed firmware, bootloader and OS can be booted. It provides users with the opportunity to run the software of their choice in the most secure manner.

Other end user impact

Users need to use properly signed deploy and boot components. Currently pxe_ilo driver would support deploy and boot images having shim and grub2 signed by Linux OS vendors. If user wants to use custom signed images, then he would need to manually configure their keys to UEFI using HP Proliant tools. If user were to use an unsigned image for deploy with flavor requesting UEFI secure boot, then deploy process would go through successfully, but final boot into instance image would fail. The signature validation of unsigned components would fail resulting in the failure of boot process. The appropriate boot failure message would get displayed on Nodes console.

Scalability impact

None

Performance Impact

There is no performance impact due to signature validation in secure boot.

Other deployer impact

User can deploy only signed images with UEFI secure boot enabled.

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
Shivanand Tendulker (stendulker@gmail.com)

Work Items

1. Add support for grub2/shim as a alternate UEFI bootloaders for Ironic pxe driver.
2. Implement secure boot for pxe_ilo driver.

Dependencies

Signed user images. The necessary DiskImageBuilder changes has been done to build signed Ubuntu and Fedora images.

Testing

Unit tests would be added for all newly added code.

Upgrades and Backwards Compatibility

None. grub2 would be alternate bootloader, which user can use only if it needs UEFI secure boot functionality.

Documentation Impact

Newly added functionality would be appropriately documented.

References

1. UEFI specification http://www.uefi.org
2. Proliantutils module - https://pypi.python.org/pypi/proliantutils
3. HP UEFI System Utilities User Guide - http://www.hp.com/ctg/Manual/c04398276.pdf

4. Secure Boot for Linux on HP Proliant servers http://h20195.www2.hp.com/V2/getpdf.aspx/4AA5-4496ENW.pdf

Client Caching Of Negotiated Version

https://bugs.launchpad.net/ironic/+bug/1526411

This adds support for caching the version negotiated by the ironicclient, between itself and the ironic server. This is supplementary to the api microversion spec approved in the Kilo release[0].

Problem description

When the ironicclient talks to the ironic server, there may be a mismatch in supported API versions. The client and server negotiate a version to be used in communicating, but since the ironicclient can be used as a user interactive client, this process of negotiation would be repeated for each command line invocation.

It would be useful, for each ironic server that the ironicclient talks to, to cache the version agreed upon for communication, so that each conversation between client and server does not require renegotiation.

This version caching would need to be per user, for each ironic server (host, network port pair specific) and would be time-bound (say 5 minutes), so any future upgrade of the client or server would benefit from supporting newer available versions.

Proposed change

The proposed implementation consists of caching the negotiated version information between an ironic-client and ironic server in local file storage for use by future invocations of the ironicclient.

This information would be cached for a specific period of time, before becoming stale and ignored.

Specifically, we are proposing:

- Using the dogpile.cache[1] caching system, a pre-existing library that is already included in global requirements[2]. It is currently used by os-client-config[3], which is used by openstackclient[4].
- Storing the cached information in local file storage, using appdirs[5] to provide the correct location
- Indexing the version information in an ironic-server:network-port pair (such as example.com:1234) so that multiple ironic servers running on the same IP address will be cached independently for each user invoking python-ironicclient
- Having a default period of 5 minutes for caching version information for each ironic server

Storing the version information in a well-known, standardised, local file storage location means that, if the user wants to, they can remove the cached version information manually triggering a renegotiation of the version to be used in communication between the client and server.

Alternatives

An alternative file-based solution was proposed[6], but rejected in favour of using dogpile.cache.

The suggestion to move to dogpile.cache was made both in the code review[9] and was discussed in IRC[8].

Reasons for using dogpile.cache included: commonality with existing file caching libraries used elsewhere in OpenStack, and use of tested common libraries typically means less bugs.

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

This spec only affects the python-ironicclient, not ironic server.

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

N/A

Security impact

None

Other end user impact

None

Scalability impact

This change potentially reduces network traffic between the client and server and hence aids scalability.

Performance Impact

This change potentially reduces network traffic between the client and server and hence improves latency between when a request is made to ironic and when the response is received.

Other deployer impact

None

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
mrda - Michael Davies <michael@the-davies.net>

Work Items

- The implementation of this spec has already commenced - see [7]

Dependencies

None

Testing

Unit tests will be provided to verify this solution

Upgrades and Backwards Compatibility

None

Documentation Impact

None

References

- [0] API Microversions Spec: http://specs.openstack.org/openstack/ironic-specs/specs/kilo/api-microversions.html
- [1] Documentation on dogpile.cache is found here: https://dogpilecache.readthedocs.org/en/latest/
- [2] dogpile.cache is already specified in https://github.com/openstack/requirements/blob/master/global-requirements.txt
- [3] https://github.com/openstack/os-client-config
- [4] https://github.com/openstack/python-openstackclient
- [5] Documentation on appdirs is found here: https://pypi.org/project/appdirs
- [6] Original custom file cache solution: https://review.opendev.org/#/c/173674/1/
- [7] Current state of the implementation at the time of this spec being raised: https://review.opendev.org/#/c/173674/19

- [8] http://eavesdrop.openstack.org/irclogs/%23openstack-ironic/%23openstack-ironic.2015-05-11.log.html#t2015-05-11T19:38:04
- [9] https://review.opendev.org/#/c/173674/9

Wake-On-Lan (WOL) power driver

https://blueprints.launchpad.net/ironic/+spec/wol-power-driver

This blueprint adds support for Wake-On-Lan (WOL) power interface in Ironic.

Problem description

Wake-On-Lan is a standard that allows a computer to be powered on by a network message. This is widely available and doesnt require any fancy hardware to work with. This is useful for users that want to try Ironic with real bare metal instead of virtual machines and only have some old PCs around.

Proposed change

The proposed implementation consists of a driver power interface implementation that will create and send Wake-On-Lan magic packets for each port (MAC addresses) registered on the node resource in Ironic.

Its important to note that Wake-On-Lan is only capable of powering **on** the machine. After the machine is unprovisioned it needs to powered off manually.

Driver properties that can be specified by the user::

wol_host: Broadcast IP address to send the magic packets. Defaults to `255.255.255.255`.

wol_port: Destination port to send the magic packets; defaults to 9.

When powering **off** is called we are just going to log a message saying the operation isnt supported by the driver and require manual intervention to be performed.

When **reboot** is called the driver will try to power **on** the machine.

When getting the power state of the node, the driver will rely on whatever is in the Ironic database and return it, since we dont have any reliable way of knowing the current power state of the machine.

Alternatives

A alternative would be relying on some external mechanism to power control the nodes. Such as `iBoot` which Ironic already have a driver for. But for that the user will need to spend some money (~200 USD) to buy an `iBoot` device.

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Security impact

This spec only covers the bits of sending a magic packet **without** the `SecureOn` feature.

`SecureOn` allows a client to append a password to the magic packet so NICs that support the feature will check prior to powering on the machine and if the MAC address + password are correct only then the system is awake . This is good against `brute force attacks`, but will be left for future work.

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

None

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
lucasagomes

Other contributors:
None

Work Items

- Write the Wake-On-Lan power interface
- Write unittests

Dependencies

None

Testing

Unittests will be added as part of the work.

Upgrades and Backwards Compatibility

None

Documentation Impact

The driver should be documented under http://docs.openstack.org/developer/ironic/index.html. The documentation will also be clear about the use of this driver, this is a testing driver and not meant for production use.

References

- Wake-On-Lan: http://en.wikipedia.org/wiki/Wake-on-LAN

5.14 Kilo

5.14.1 AMT PXE Driver

https://blueprints.launchpad.net/ironic/+spec/amt-pxe-driver

This blueprint implements a new driver PXEAndAMTDriver which supports deployment for AMT/vPro system on Desktops.

Problem description

Currently there is no support with Ironic to do deployment for Desktops within AMT/vPro system. This BP will extend Ironic to Desktop area.

Proposed change

Implement a new driver PXEAndAMTDriver that uses amt to control the power of nodes with AMT System and uses pxe to deliver the image to nodes. Following are details,

- Add new class PXEAndAMTDriver inherited from base.BaseDriver in ironic/drivers/pxe.py
- Add new class AMTPower inherited from base.PowerInterface in ironic/drivers/modules/amt/power.py

  - `validate()` - Validate the node driver info
  - `get_power_state()` - Get the power state from the node
  - `set_power_state()` - Set the power state of the node, such as power on/off
  - `reboot()` - reboot the node

- Add new class AMTMangement inherited from base.ManagementInterface in ironic/drivers/modules/amt/management.py
  - `validate()` - Validate the node driver info
  - `ensure_next_boot_device()` - ensure the next boot device of the node

    Note

    AMT/vPro only accept the first boot device and ignore the rest if we send multiple _set_boot_device_order requests to AMT nodes. For example, when the user set boot device twice, the node will boot with the first one. So AMT driver only save amt_boot_device into DB via set_boot_device() and send the request to the node via ensure_next_boot_device() before set the node power on. So that AMT driver can support users to set boot device multiple times like other drivers.

  - `set_boot_device()` - Set the boot device of the node.

    Note

    As AMT/vPro doesnt support set boot device persistent in BM node like BMC, it only set boot device for one time. So AMT driver call ensure_next_boot_device() between every power cycle if boot device is persistent. AMT driver saves amt_boot_device/amt_boot_persistent into node.driver_internal_info, which will be read by ensure_next_boot_device().

  - `get_boot_device()` - Get the boot device of the node
- Add a condition to enable AMT driver to call ensure_next_boot_device in pxe._continue_deploy

  Note

  During PXE deploy processing, after finish dd, the target machine will reboot by ramdisk rather than Ironic. AMT Driver has to call ensure_next_boot_device again in _continue_deploy().

Alternatives

- **Save amt_boot_device and amt_boot_persistent in:**
  - driver_info but user will aware the change of boot device and could be different from his input.
  - extra but not for use from inside Ironic.

Data model impact

None

REST API impact

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

The following driver_info fields are required:

- amt_address: hostname or IP of AMT node
- amt_password: password used for connect to AMT node
- amt_username: username used for connect to AMT node
- amt_protocol: protocol used for connect to AMT node (optional)

The following parameters are added into newly created [amt] section in ironic.conf.

- protocol: default value of AMT (http/https) protocol. The default value is http
- max_retry: default retries for AMT power operations. The default value is 3 times.
- action_wait: default seconds for driver to wait for retries. The default value is 10 seconds.

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
tan-lin-good

Work Items

Implement `PXEAndAMTDriver` class inherited from `base.BaseDriver`.

Implement `AMTPower` class inherited from `base.PowerInterface`

Implement `AMTManagement` class inherited from `base.managementInterface`

Dependencies

openwsman-python package

> **Note**
>
> AMT deprecated SOAP (amttool) support after the latest version 9.0. http://en.wikipedia.org/wiki/Intel_AMT_versions Intel AMT 9.0 SOAP(EOI) protocol removed. So AMT only support WS-MAN protocol (openwsman) now. The solution with openwsman works for AMT 7.0/8.0/9.0. AMT 7.0 is released in 2010, so most PCs with vPro are involved.

Testing

Will add Unit Testing.

Upgrades and Backwards Compatibility

None

Documentation Impact

Will document the usage of this driver.

References

None

5.14.2 Ironic Microversions

https://blueprints.launchpad.net/ironic/+spec/api-microversions

The purpose of this spec is to call out the specific behaviour between Ironic and python-ironicclient that is required now that we are using microversions, and to provide guidance how other clients may wish to interact with Ironic.

Problem description

As a community we are really good at evolving interfaces and code over time via incremental development. Weve been less good at giant big bang drops of code. The Ironic API is under heavy development, and we want to ensure that consumers of Ironics API are able to make use of new features as they become available, while also ensuring they dont break due to incompatibilities.

Ironic isnt the only OpenStack project implementing microversions, ironically. Nova[0] is also implementing microversions in parallel for the Kilo release.

The implementation of microversions is currently underway in Ironic, based around that Nova spec. The implementation in Ironic has already landed[1], and is currently under implementation for Ironics primary client, python-ironicclient[2].

Microversions are implemented in the API through the addition of a new HTTP header - specifically X-OpenStack-Ironic-API-Version. This header is accepted by Ironic so a client can indicate which version of the API it wants to use for communication, and likewise for Ironic to indicate which version it is using for communication.

For Ironic, if no HTTP header is supplied, v1.0 (stable/juno) of the API is used. If an invalid version is specified in the HTTP header, an HTTP 406 Not Acceptable is returned with an indication of the supported range of versions[4]. If the special latest version is specified, Ironic will use its most recent version.

During changes being made to python-ironicclient[2] to support Ironics microversions it was discovered that there isnt a formal specification of how Ironic and a client should interact for varying cases of microversion mismatch or for an unknown/unspecified microversion.

The need for this spec was discussed in IRC[3], to specifically address the interaction between Ironic and a client (whether that be python-ironicclient or any other client).

Proposed change

To address the specific behaviour between Ironic and python-ironicclient, the following Use Cases are listed to specify the expected functionality. Please see the IRC logs[3][5] and this code review[2] for further context for these Use Cases.

For the purposes of definition, we will use the term old Ironic to refer to a version of Ironic that predates microversions and has no knowledge of them. Likewise, we will use the term new Ironic to refer to a version of Ironic that includes support for microversions.

Similarly, we will apply such labelling to old client and new client respectively.

New Client Default Microversion

The python-ironicclient provides an option to specify which microversion to attempt to use for communicating to Ironic. When this is specified, the requested microversion should be used (unless of course the client cannot support that version). This includes latest which is an indication to Ironic that the latest microversion that ironic knows about should be used.

However if no microversion is specified to the client, it should use the latest microversion that the client supports.

As part of the initial communication with Ironic it may need to revert to a lower microversion due to Ironics support for only older microversions (as per Use Case 7).

The goal of this requirement is for python-ironicclient / Ironic communication to just work for the user, and if possible, to use the most recent version of the REST API possible, so that the user is able to make use of the latest functionality.

Use Case 1: Old Client communicating with an Old Ironic

This is exactly the same behaviour that was seen prior to the introduction of microversions - no change to either the client or server is required for this case.

- The client makes a connection to Ironic, not specifying the HTTP header X-OpenStack-Ironic-API-Version
- Ironic does not check for an X-OpenStackIronic-API-Version header, and processes all communication simply as v1.0 (stable/juno)

Use Case 2: Old Client communicating with a New Ironic

This is where Ironic is updated to a new version that support microversions, but an old client is used to communicate to it.

- The client makes a connection to Ironic, not specifying the HTTP header X-OpenStack-Ironic-API-Version
- Ironic does not see the X-OpenStack-Ironic-API-Version HTTP header
- Ironic communicates using v1.0 (stable/juno) of the REST API, and all communication with the client uses that version of the interface.

Use Case 3A: New Client communicating with a Old Ironic (not user-specified)

This is the where the user does not request a particular microversion to a new client that support microversions and tries to communicate with an old Ironic. The version that the new client uses is the maximum microversion it supports.

- The user does not specify the microversion to use in communication with the client. Consequentally, the client attempts to use the latest microversion that the client knows about.
- The client makes a connection to an old Ironic, supplying a X-OpenStack-Ironic-API-Version HTTP header
- Ironic doesnt look for, or parse the HTTP header. It communicates using the only API code path it knows about, that being v1.0
- The client does not receive a X-OpenStack-Ironic-API-Version header in the response, and from that is able to assume that the version of Ironic that it is talking to does not support microversions. That is, it is using a version of the REST API that predates v1.0 (stable/juno).
- The client should transparently proceed now that it knows that it is communicating to an Ironic that can only support v1.0 of the REST API.

Use Case 3B: New Client communicating with a Old Ironic (user-specified)

This is the where the user requests a particular microversion to a new client that support microversions and tries to communicate with an old Ironic.

- The user specifies a microversion that is valid for the client.

- The client makes a connection to an old Ironic, supplying a X-OpenStack-Ironic-API-Version HTTP header
- Ironic doesnt look for, or parse the HTTP header. It communicates using the only API code path it knows about, that being v1.0
- The client does not receive a X-OpenStack-Ironic-API-Version header in the response, and from that is able to assume that the version of Ironic that it is talking to does not support microversions. That is, it is using a version of the REST API that predates v1.0.
- The client informs the user that it cannot communicate to Ironic using that microversion and exits.

Use Case 4: New Client, user specifying an invalid version number

This is the case where a user provides as input to a new client an invalid microversion identifier, such as spam, l33t, or 1.2.3.4.5.

- The user specifies a microversion to the client that is invalid. The client should return an error to the user, i.e. the client should provide some validation that a valid microversion identifier is provided.

Use Case 5: New Client/New Ironic: Unsupported Ironic version

This is the case where a new client requests a version that is older than the new Ironic can handle. For example, the client supports microversions 1.1 to 1.6, and Ironic supports versions 1.8 to 1.15.

- The client makes a connection to Ironic, supplying 1.6 as the requested microversion.
- Ironic responds with a 406 Not Acceptable, along with the -Min- and -Max- headers that it can support (in this case 1.8 and 1.15)
- As the client does not support a version supported by Ironic, it cannot continue and reports such to the user.
- (An alternative path would be for the client to try and proceed using a version acceptable to Ironic. Note that in this case the client should be able to proceed since any change that would break basic compatibility would likely require a major version bump to v2)

Use Case 6: New Client/New Ironic: Unsupported Client version

This is the case where a new client requests a version that is newer than the new Ironic can handle. For example, the client supports microversions 1.10 to 1.15, and Ironic supports versions 1.1 to 1.5.

- The client makes a connection to Ironic, supplying 1.10 as the requested microversion.
- Ironic responds with a 406 Not Acceptable, along with the -Min- and -Max- headers that it can support (in this case 1.1 and 1.5)
- The client reports this error to the user
- (An alternative path would be for the client to try and proceed using a version acceptable to Ironic. Note that in this case the client should be able to proceed since any change that would break basic compatibility would likely require a major version bump to v2)

Note: This scenario should not occur in practice as the client should always be able to talk to any version of Ironic.

Use Case 7A: New Client/New Ironic: Negotiated version (not user-specified)

This is the case where a new client requests a version that is newer than the new Ironic can handle, but supports a version that Ironic supports. For example, the client supports microversions 1.8 to 1.15, and Ironic supports versions 1.1 to 1.10.

- The user has not specified a version to the client
- The client makes a connection to Ironic, supplying 1.15 as the microversion since this is the latest microversion that the client supports.
- Ironic responds with a 406 Not Acceptable, along with the -Min- and -Max- headers that it can support (in this case 1.1 and 1.10)
- The client should transparently proceed, having negotiated that both client and server will use v1.10. The client should also cache this microversion, so that subsequent attempts do not need to renegotiate microversions.

Use Case 7B: New Client/New Ironic: Negotiated version (user-specified)

This is a slight variation on Use Case 7A, where the user specifies a specific version to use to communicate with Ironic.

- The user specifies a particular microversion (e.g. 1.15) that the client should use
- The client makes a connection to Ironic, supplying 1.15 as the microversion
- Ironic responds with a 406 Not Acceptable, along with the -Min- and -Max- headers that it can support (in this case 1.1 and 1.10)
- The client reports this to the user and exits

Use Case 8: New Client/New Ironic: Compatible Version

This is the case where a new client requests a version that is supported by the new Ironic. For example, the client supports microversions 1.8 to 1.10, and Ironic supports versions 1.1 to 1.12.

- The client makes a connection to Ironic, supplying 1.10 as the requested microversion.
- As Ironic can support this microversion, it responds by sending back a response of 1.10 in the X-OpenStack-Ironic-API-Version HTTP header.

Use Case 9: New Client/New Ironic: Version request of latest

This is the case where a new client requests a version of latest from a new Ironic.

- The client makes a connection to Ironic, supplying latest as the version in the X-OpenStack-Ironic-API-Version HTTP header
- Ironic responds by using the latest API version it supports, and includes this in the X-OpenStack-Ironic-API-Version header, along with the -Min- and -Max- headers.

Note: Its possible that Ironic provides a response that the client is not able to correctly interpret. This is unavoidable, however it enables a client that is older than the deployed version of Ironic to potentially access all of the functionality available in that Ironic version. In this instance, the client may choose to report to the user the version that Ironic included in the response, along with the min and max microversions that the client is known to be able to support. Any parts of the response from Ironic that the client is not programmed to handle will simply be discarded.

Alternatives

One alternative to microversions is to not have them at all. What this would result in would be a group of large changes happening simultaneously, resulting in unpaired server/client versions not being compatible at all. It would also result in less frequent, but larger incompatible API changes. And nobody wants that.

Data model impact

None. This change is isolated to the API code.

REST API impact

As described above, a new HTTP header would be accepted, and returned by Ironic.

If a client chose to use that header to request a specific version, Ironic would respond, either accepting the requested version for future communication, or rejecting that version request as not being supportable.

If a client chose not to use that header, Ironic would assume that the REST API to be used would be v1.0 (that is, the same API that was present in the Juno release[6]). This is how the REST API works today.

RPC API impact

None

Driver API impact

None

Nova driver impact

The current behaviour of python-ironicclient (pass no version header) results in the Nova driver using v1.0 of our API. The proposed changes to python-ironicclient will cause the Nova driver to use the latest microversion that the client supports. This will make available to Nova any new functionality we add to Ironic at the point in time when we tag a new client release.

A future enhancement would be to modify the Nova Ironic driver to specify a specified microversion to use when communicating to Ironic. This would provide exact control over which REST API version to consume.

This behaviour should be documented in how Nova and Ironic are gate tested.

There is the potential here to break the nova driver if the incorrect version is requested. Consequently, it is important to manage the Nova driver, Ironic and python-ironicclient version changes.

Security impact

None

Other end user impact

Clients that wish to use new features available over the REST API added since the Juno release will need to start using this HTTP header. The fact that new features will only be added in new versions will encourage them to do so.

Scalability impact

None

Performance Impact

None

Other deployer impact

None

Developer impact

Any future changes to Ironics REST API (whether that be in the request or any response) *must* result in a microversion update, and guarded in the code appropriately.

Upgrades and Backwards Compatibility

As described above.

Implementation

Assignee(s)

Primary assignees:

```
lintan - Tan Lin <tan.lin.good@gmail.com>
```

Secondary assignees:

```
devananda - Devananda van der Veen <devananda.vdv@gmail.com>
rloo - Ruby Loo <rloo@yahoo-inc.com>
mrda - Michael Davies <michael@the-davies.net>
plus many others
```

Work Items

Complete the python-ironicclient microversion implementation by:

1. Add in the highest Ironic microversion that the python-ironicclient can support.
2. If the User does not pass a version, the client should automatically try the highest version it supports. That is, send the X-OpenStack-Ironic-API-Version HTTP header with the highest Ironic microversion that it supports.
3. The python-ironicclient should support X.Y and latest as valid API versions.

Dependencies

None

Testing

It is not feasible for tempest to test all possible combinations of the API supported by microversions. We will have to pick specific versions which are representative of what is implemented. The existing tempest tests will be used as the baseline for future API version testing.

The following combinations should be tested:

- Old client (eg, juno-era client release) against current master branch of Ironic
- Latest client (eg, proposed changes to master) against current master branch of Ironic
- Latest client (eg, proposed changes to master) against stable/juno Ironic

And we should continue such forwards-and-backwards testing for as long as we claim to support a given release.

Documentation Impact

No specific documentation impact is identified that is not covered by existing API change processes.

References

- [0] http://specs.openstack.org/openstack/nova-specs/specs/kilo/implemented/api-microversions.html for details on Novas microversioning. Note that this document borrows heavily from that spec. (Thanks cyeoh!)
- [1] https://review.openstack.org/#/c/150821/ and https://review.openstack.org/#/c/158601/
- [2] https://review.openstack.org/#/c/155624/
- [3] http://eavesdrop.openstack.org/irclogs/%23openstack-ironic/%23openstack-ironic.2015-03-03.log#2015-03-03T22:17:26-2015-03-03T23:00:42
- [4] https://review.openstack.org/#/c/160758/
- [5] http://eavesdrop.openstack.org/meetings/ironic/2015/ironic.2015-03-09-17.00.log.txt#17:17:33-17:50:20
- [6] While this is broadly true, at least one change - the addition of the maintenance_reason field - has been made since the Juno release of Ironic

5.14.3 automate-uefi-bios-iso-creation

https://blueprints.launchpad.net/ironic/+spec/automate-uefi-bios-iso-creation

This spec proposes to add support creation of dual(bios+uefi) boot ISO automatically.

Problem description

Today, a dual-mode boot ISO has to be created manually, uploaded to Glance, and associated with the instance image in Glance by setting its boot_iso image property. The deploy image can be modified using GRUB on runtime. This is required by the iscsi_ilo driver to enable UEFI boot mode.

Proposed change

This spec proposes to use GRUB to create the dual-mode boot ISO on the fly in ironic/common/images.py. This image will be uploaded to swift and associated with the node for which it was created, and will not be shared between instances, even when using the same instance image. When the instance is deleted, the boot_iso file is deleted from swift. The steps would be as follows:

1. Create isolinux config file.
2. Create the grub config files.
3. Create the bootx64.efi.
4. create vfat image efiboot.img
5. Create ISO using mkisofs with the following options added -eltorito-alt-boot -e isolinux/efiboot.img to existing syntax.
6. The boot iso is created by the driver and uploaded at the swift container.

when node is torn down, the boot iso is deleted by the driver.

Alternatives

To boot up in UEFI mode, the deployer can create the ISO manually using disk-image-builder utility disk-image-create and upload on glance.

Data model impact

None.

REST API impact

None.

RPC API impact

None.

Driver API impact

None.

Nova driver impact

None.

Security impact

None.

Other end user impact

None.

Scalability impact

None.

Performance Impact

None.

Other deployer impact

The boot iso can be created and uploaded to the swift on the fly when deploy is invoked for iscsi_ilo driver i.e. current manual step for updating the deploy image with the boot iso will not be required after the enhancement.

Developer impact

None.

Implementation

Assignee(s)

Primary assignee:
agarwalnisha1980

Work Items

In /opt/stack/ironic/ironic/common/images.py file,

- To create the conf file for UEFI
- To enhance the function create_isolinux_image() for creating UEFI capable ISO.
- To remove the check in /opt/stack/ironic/ironic/drivers/modules/ilo/deploy.py for erroring out when the deploy image doesnt has the boot_iso uuid updated.

Dependencies

None.

Testing

The unit tests will be added/updated as required for the code.

Upgrades and Backwards Compatibility

The iscsi_ilo driver will support both the manual and automated boot iso methods. By default, if the image already has the boot_iso property populated, the driver will consume the boot_iso from it.

Documentation Impact

None.

References

None.

5.14.4 Driver Internal Info

https://blueprints.launchpad.net/ironic/+spec/driver-internal-info

Problem description

Driver should have its own infos which cannot manipulated by user/admin. These infos are not input from admin like driver_info and they may vary during the deployment process. They can only be used by driver itself.

One example is ipmitool. Not all IPMI firmware support set boot device persistent, so we need to save this locally.

Proposed change

- Add a new internal attribute *driver_internal_info* in nodes table, which cannot modify by Admin/user by calling node.update API
- Modify node.update to clear *driver_internal_info* when update driver via node.update API.

Alternatives

Saving these infos in a new table named driver_interal_info.

Data model impact

Add a new internal attribute *driver_internal_info* in *node* table. This field is a json dict.

REST API impact

The *driver_internal_info* field should be added to the node details API.

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

None

Developer impact

Other drivers should save their own infos into the new attribute.

Implementation

Assignee(s)

Primary assignee:
tan-lin-good

Work Items

- Add *driver_internal_info* to the nodes table with a migration.
- Update object Node.
- Support clean a nodes driver_internal_infos when it changes its driver.
- Update some drivers with this feature.

Dependencies

None

Testing

Add unit tests.

Upgrades and Backwards Compatibility

Add a migration script for DB.

Documentation Impact

Update the developer documentation.

References

None

5.14.5 Allow drivers to have their own periodic tasks

https://blueprints.launchpad.net/ironic/+spec/driver-periodic-tasks

This spec suggests allowing Ironic drivers to define their own periodic tasks.

Problem description

Currently Ironic conductor can run periodic tasks in a green thread. However, if some driver requires a driver-specific task to be run, it needs to patch conductor manager, which is not acceptable.

Proposed use cases:

- For in-band inspection using discoverd driver-specific periodic task would be used to poll discoverd for inspection status.
- For DRAC RAID implementation driver-specific periodic task may be used again to poll status information from BMC.
- For deploy drivers supporting long-running ramdisks (e.g. IPA) a driver-specific periodic task may be used to poll for dead ramdisks when nothing is deployed on a node.

Proposed change

- Create a new decorator `@ironic.drivers.base.driver_periodic_task` to mark driver-specific periodic tasks. It will mostly delegate its job to `@periodic_task.periodic_task` with the exception of additional argument `parallel` defaulting to `True`.

  Until parallel periodic tasks is implemented in Oslo, `parallel=True` will be implemented by wrapping the task into a function calling `eventlet.greenthread.spawn_n` to make it run in a new thread. It will also use Eventlet semaphore to prevent several instances of the same task from running simultaneously.

- Modify `ConductorManager.__init__` to collect periodic tasks from each present interface of each driver. It should use existing markers added by `@periodic_task.periodic_task` to a method to detect periodic task. Information about a periodic tasks should be placed in `_periodic_spacing` `_periodic_last_run` and `_periodic_tasks` attributes of the conductor.

- The future modification should be adding something like `add_periodic_task` to a `PeriodicTasks` class in Oslo. The only thing that prevents me from suggesting it is that `PeriodicTasks` class is under graduation into a new oslo.service now. No changes are allowed at the moment. This should be done as a refactoring step later.

- Once `oslo.service` is graduated and parallel periodic tasks are implemented there, get rid of the work around inside `driver_periodic_task`, and switch to using parallel periodic tasks from Oslo.

Alternatives

- Each time modify conductor when we need a periodic task. That requires a consensus on how to make it in a generic way.
- Just use `LoopingCall`. I believe that this approach is less controllable (in terms of how many threads we run, how many requests we make to e.g. DRAC BMC etc) and leads to code duplication.

Data model impact

None

REST API impact

None

RPC API impact

None

Driver API impact

No impact for the driver API itself.

Nova driver impact

None

Security impact

None

Other end user impact

None

Scalability impact

None expected

Performance Impact

This will allow large number of periodic tasks to be added to Ironic. Defaulting to `parallel=True` should minimize effect on performance. The decision of whether to add or not a new tasks is anyway to be done on a case-by-case basis.

Other deployer impact

None.

Note that periodic_max_workers and rpc_thread_pool_size configuration options are not affecting driver-specific periodic tasks.

Developer impact

Driver developers can mark any method as a `@driver_periodic_task`.

Implementation

Assignee(s)

Primary assignee:
Dmitry Tantsur, LP: divius, IRC: dtantsur

Work Items

- Add `ironic.drivers.base.driver_periodic_task` decorator
- Modify `ConductorManager.__init__`.
- Propose `add_periodic_task` to `oslo.service`.

Dependencies

- parallel periodic tasks are nice to have, but is not a hard dependency.

Testing

Unit testing will be conducted.

Upgrades and Backwards Compatibility

None

Documentation Impact

Update driver interface documentation to mention how to create periodic tasks.

References

Parallel periodic tasks spec: https://review.openstack.org/#/c/134303/

5.14.6 Expose configdrive to instances

https://blueprints.launchpad.net/ironic/+spec/expose-configdrive

This blueprint adds support for exposing a configdrive image[1] to instances deployed by Ironic.

Problem description

Instances deployed by Ironic should be able to use cloud-init (or similar software) to put an end users data on an instance. This is possible today with Ironic by including cloud-init with the image, and pointing it at a Nova metadata service.

There are two issues with this approach:

- Some deployers do not run a metadata service in their environment.
- If a deployer provisions Ironic machines using static IP address assignment, the instance will not have network access until cloud-init puts the network configuration into place. If the metadata service is the only way to get the network configuration, the instance is deadlocked on getting network access.

To solve these problems, a configdrive image can take the place of the metadata service. In the VM world, this is typically handled by the hypervisor exposing a configdrive image to the VM as a volume.

In Ironics case, there is no hypervisor, so this image needs to be exposed to the instance in some other fashion. This could be accomplished by writing the image to a partition on the node, exposing the image via the out-of-band mechanism (e.g. a virtual floppy in HPs iLO), or configuring the node to mount the image from a SAN. In any case, this needs to be handled by Ironic, rather than Nova. However, Nova has the data that belongs in the configdrive, as well as the code to generate the image. So, it makes sense for Nova to generate an image and pass it to Ironic.

This blueprint outlines how Ironic will handle the configdrive image that Nova provides and expose it to an instance.

Proposed change

Novas Ironic virt driver will generate a config drive image, gzip and base64 encode it and pass to the Ironic service as part of the setting provision state call. This is discussed in more detail in this nova spec.[0]

For that, we have to extend our API to optionally accept a config drive as part of the request BODY.

If the config drive is present, Ironic will either upload the data to Swift and update the Nodes instance_info to include the temporary URL from the upload or if swift if not configured, the config drive data will be stored directly into the Nodes instance_info field.

From there deploy drivers will be responsible for exposing the configdrive to the instance, as well as removing the configdrive from the instance upon deletion. This cannot be coordinated by code outside of the driver, as only the driver can know when and how to take these actions.

Some mechanisms a driver may use to expose a configdrive include:

- Write the image to the instances disk as a partition.
- Use an OOB mechanism to mount the image as a virtual disk.
- Use an OOB mechanism to configure the instance to mount the image from a SAN.

Alternatives

There are no clear alternatives here.

Data model impact

A configdrive key will be added to node.instance_info.

REST API impact

Extend the /nodes/<uuid>/provision endpoint to accept an optional `configdrive` parameter as part of the request BODY.

Since the config drive is only valid when spawning an instance, in the Ironic API passing the `configdrive` parameter will be only valid when setting the Nodes provision state to `active`. Passing the parameter to any other provision state should return HTTP 400 (Bad Request).

RPC API impact

The config drive passed via the API should be passed down to the `do_node_deploy` RPC method.

Driver API impact

None.

Nova driver impact

The nova driver will need to implement the functionality to generate the configdrive image and get it to Ironic.

This is discussed in the corresponding nova spec.[0]

Security impact

This proposal involves storing end user data as an image in Swift. This may be a security concern, as this data is not encrypted at rest.

There are methods for securing this data that are out of the scope of this initial work.

Other end user impact

None.

Scalability impact

Use of the configdrive would require a call to Swift (or some other object store), which will have some impact on the system, but is probably negligible.

Performance Impact

None.

Other deployer impact

This feature might require deploying some object store service (Swift for the reference implementation).

Developer impact

Developers writing drivers should implement this functionality, but it is not required, as it may not be possible for some drivers.

Implementation

Assignee(s)

Primary assignee:
lucasagomes <lucasagomes@gmail.com>

Other contributors:
jroll <jim@jimrollenhagen.com>

Work Items

- Implement the Nova side of this feature.
- Implement functionality for various deploy drivers.
- Add support to IPA to fetch a configdrive by URL. It currently only supports being passed a blob in the prepare_image command.
- Add support to PXE drivers to consume the config drive and expose it to the tenants.
- Add tempest tests (in conjunction with the Nova driver).

Dependencies

This change depends on the corresponding Nova spec.[0]

Testing

A tempest test should be added that deploys a bare metal instance with a configdrive, and verifies that the configdrive is properly written to the instance.

Upgrades and Backwards Compatibility

The Ironic code will need to be deployed before enabling configdrive support in Nova.

This feature is completely optional, so it is backward compatible.

Documentation Impact

Documentation may need to be updated to indicate that a configdrive may be used with instances deployed by Ironic.

References

[0] https://blueprints.launchpad.net/nova/+spec/use-configdrive-with-ironic

[1] http://cloudinit.readthedocs.org/en/latest/topics/datasources.html#config-drive

5.14.7 Extend Vendor Passthru

https://blueprints.launchpad.net/ironic/+spec/extend-vendor-passthru

Make the VendorPassthru and DriverVendorPassthru more consistent and extend it to support running vendor methods synchronously and asynchronously as well as different HTTP methods apart from POST.

Problem description

The VendorPassthru and DriverVendorPassthru are inconsistent. VendorPassthru methods are asynchronous and DriverVendorPassthru methods are synchronous.

In addition, right now each driver is responsible for implementing their own method that will invoke their vendor methods and they ended up duplicating a lot of code and also doing things like raising different exceptions for the same type of error. Also, some of them are logging the errors raised by the vendor methods or converting non-Ironic exceptions to Ironic exceptions and others are not.

Apart from that both endpoints only support one HTTP method: POST. It would be good to extend it to support other HTTP method because some drivers may want to use them. One example would be the

iPXE driver where supporting GET would be beneficial because the iPXE script could request a GET in the Ironic API which would return what kernel and ramdisk should be booted and the script can chainload it from there.

Another problem is that currently the vendor methods are not discoverable via out API, theres no way to know what are the methods being currently exposed by the drivers.

Proposed change

The first change is to have a generic way to run the vendor methods, and this should be responsible for logging and converting exceptions in case a non-Ironic exception has been raised. The way this blueprint wants to do that is by creating a decorator to decorate the vendor methods and wrap it with a custom code that will take care of those.

The second part will extend the new decorator to add some metadata about each vendor method and be able to create a generic mechanism to invoke those methods. Vendors that want to expose some unique capability in Ironic should not care about writing a custom code to invoke their functions as part of the work to implement a vendor interface, they should only care about implementing the vendor method itself. For this work we also need to differentiate VendorPassthru from DriverVendorPassthru methods so the decorator should be split in two, but they both can share the same code base, it will be more a alias for each type.

The third part is about making VendorPassthru and DriverVendorPassthru more consistent. We should make it configurable per-method if it should run asynchronously or synchronously instead of having each endpoint running things in a different way. A flag will be added to the decorators to them express that.

The fourth will is about allowing vendor methods to support different HTTP methods. Since its a REST API vendors would benefit from being able to expose methods to use all HTTP methods available. We should though inform the vendors that things like having a synchronous GET method talking to the BMC may not be a good idea since BMCs are flaky and it may take a long time for the request to complete, so they dont shoot themselves on the foot. But for things like the iPXE driver generating configurations on-the-fly that would be beneficial.

Once we have the generic mechanism mapping all vendor methods, (the second part of the proposal) we can then expose those via our API. By issuing a GET on ../vendor_passhru/methods endpoint it will return the available methods for that driver or node. For each method, it will include a description, the supported HTTP methods and whether it is a synchronous or asynchronous operation.

Another change would include a tutorial about how to write vendor methods, and give tips just like the one about not talking to the BMC in a synchronous way.

A backward compatible layer will also be added so we dont break out-of-tree drivers that still use a custom vendor_passthru() and driver_vendor_passthru() on their vendor interfaces. In case these methods are present in the vendor interface they will be called just like before. If not present, we are going to use the new mechanism to invoke the vendor methods.

Alternatives

- Modify the vendor_passthru() and driver_vendor_passthru() in the VendorInterface base class to run the vendor methods, basically substituting the decorator idea, that would impact on the backwards compatible plan that still need the conductor to know how it should invoke that vendor method, should it start a working thread (asynchronously) or call it directly (synchronously).

Data model impact

None.

REST API impact

- /v1/nodes/<uuid>/vendor_passthru and /v1/drivers/<driver>/vendor_passthru will support different HTTP methods apart from POST, the list of methods are: GET, POST, PUT, PATCH and DELETE.
- /v1/nodes/<uuid>/vendor_passthru will continue to return HTTP 202 (Accepted) for methods running in asynchronous mode, but will also return HTTP 200 (OK) for methods running synchronously.
- /v1/drivers/<driver>/vendor_passthru will continue to return HTTP 200 (OK) for methods running in synchronous mode, but will also return HTTP 202 (Accepted) for methods running asynchronously.
- GET /v1/nodes/<uuid>/vendor_passthru/methods will return a list of vendor methods and its metadata supported by that node.
- GET /v1/drivers/<driver>/vendor_passthru/methods will return a list of vendor methods and its metadata supported by that driver.

Note: Both endpoints already support returning error HTTP codes like HTTP 400 (BadRequest) and so on.

RPC API impact

- A new parameter *http_method* will be added to the vendor_passthru() and driver_vendor_passthru() methods in the RPC API.
- The return value from the vendor_passthru() and driver_vendor_passthru() methods in the RPC API will also change to include which mode (asynchronous or synchronous) the vendor method was invoked, thats needed so the API part can determine what HTTP code it should return in case of success.
- Two new RPC methods will be added to the RPC API: get_vendor_routes(<node id>) and get_driver_routes(<driver name>) which will return the list of available methods for that specific node or driver and will be used by the *ironic-api* services to expose this information via the REST API.

Driver API impact

- Removal of vendor_passthru() and driver_vendor_passthru() from the VendorInterface. Prior to these changes each driver had to implement their own method to invoke their vendor methods and it was done via the vendor_passthru() and driver_vendor_passthru() methods from the VendorInterface. This is going to be replaced by a common method in the ConductorManager. This also allows us to test whether drivers continue to implement a custom vendor_passthru() or driver_vendor_passthru() method in the VendorInterface and if so we invoke them to make it backward compatible. When invoked using the backward compatible mode we are going to log a warning saying that having custom vendors method has been deprecated.

Nova driver impact

None.

Security impact

None.

Other end user impact

- Support for using different HTTP methods when calling the vendor endpoints will be added in the python-ironicclient, since today it assumes POST only.

Scalability impact

None.

Performance Impact

None.

Other deployer impact

None.

Developer impact

Writing vendor methods is going to be easier and more flexible.

Implementation

Assignee(s)

Primary assignee:
lucasagomes

Other contributors:
sambetts

Work Items

- Create a decorator that will take care of logging and handling exceptions from the vendor methods.
- Extend the decorator to add metadata to the methods and be able to create a generic mechanism to invoke those without requiring vendors to write a custom code for that.
- Add support for running vendor methods synchronously and asynchronously.
- Add support for different HTTP methods on the vendors endpoints.
- Write a document explaining how to write vendor methods.
- Add client support for calling vendor methods using different HTTP methods.

Dependencies

None.

Testing

- Unittests
- Tempest tests for the API changes

Upgrades and Backwards Compatibility

This change will be backwards compatible with existing clients, so they could still run their custom vendor_passthru() and driver_vendor_passthru() methods.

Documentation Impact

- A new document will be added explaining how to write vendor methods.
- Update the Ironic documentation to mention that writing a custom vendor_passthru() and driver_vendor_passthru() methods in the vendor class has been deprecated and will be removed in the Liberty cycle.

References

None.

5.14.8 Implement Cleaning Operations for iLO drivers

https://blueprints.launchpad.net/ironic/+spec/ilo-cleaning-support

This spec proposes to support certain functionalities to be invoked as part of Cleaning process through iLO drivers to manage HP Proliant nodes.

Problem description

The spec Implement Cleaning States [1] provides a framework to invoke certain actions as part of cleaning process. Some of these operations could be hardware dependent and could be supported out-of-band via iLO.

So, iLO Drivers should be able to support Proliant specific cleaning operations.

Proposed change

- Add the following functions to IloManagement Interface and decorate the same using @clean_step(priority)
  - **ilo_reset()**
    Reset iLO.
  - **apply_base_firmware_settings()**
    To apply base firmware settings.
  - **reset_ilo_credential()**
    reset bmc password to the new one. This can be useful for the environments, where operator wants to refresh the password for the new workload.

  - **reset_secure_boot_keys()**
    To reset the secure boot keys to manufacturers defaults. (Applicable if secure boot feature is supported for ironic drivers)
  - **clear_secure_boot_keys()**
    To clear secure boot keys (Applicable if secure boot feature is supported for ironic drivers)

- **The suggested default ordering would be -**
  reset_ilo_credential() 9 reset_secure_boot_keys() 8 apply_base_firmware_settings() 7 ilo_reset() 6
- The priority of clear_secure_boot_keys() by default would be zero, operator would have option to choose either reset_secure_boot_keys or clear_secure_boot_keys(to reset to manufacturers defaults or clear all keys)
- `proliantutils` [2] library will be used for performing out-of-band operations like reset_bmc_credential, ilo_reset, apply_base_firmware_settings, reset_secure_boot_keys, clear_secure_boot_keys.
- New CONF variable CONF.ilo.base_firmware_settings would be added, which takes the default settings to be applied on cleanup of every node.
- secure_disk_erase, update_firmware will be implemented via in-band mechanism. Tools like hdparm, shred are being explored for disk erase and hpsum , smart component executables are being explored for the firmware update.
- Parameters required for cleaning operations need to be passed via driver_info with appropriate keys.

  eg: Following are the mandatory parameter keys required by the clean steps -

  1. `upgrade_firmware` - `ilo_firmware_location_url` key, which can accept the http location url or swift url for the tar/gz file of all the firmwares to be updated.
  2. `reset_ilo_credential` - `ilo_change_password`, which accepts the default iLO password to be changed during cleaning.
- If the keys are missing and if the respective clean step is enabled, warning message will be logged and the step will be no-op and continue with other clean steps.

Alternatives

None

Data model impact

None

REST API impact

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Security impact

Security is enhanced by performing some of the cleaning tasks like secure disk erase, iLO password reset etc.

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

- Deployer might need to build the IPA ramdisk with HP specific tools.
- DIB element will be enhanced to add ProLiant specific cleaning tools for deploy ramdisk. Deployer can use the enhanced DIB element to build the ramdisk.

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
ramineni

Work Items

Implement the functions that can be invoked during cleaning.

Dependencies

Depends on https://review.openstack.org/#/c/102685/ for the framework to perform cleaning.

Testing

Unit tests will be added mocking proliantutils library.

Upgrades and Backwards Compatibility

None

Documentation Impact

- Supported firmware settings will be documented.
- Parameter keys required for certain clean operations will be documented.
- Creating deploy ramdisk with HP Specific tools will be documented.

References

[1] Implement cleaning states - https://review.openstack.org/#/c/102685/

[2] proliantutils - https://github.com/hpproliant/proliantutils

5.14.9 iLO Management Interface

https://blueprints.launchpad.net/ironic/+spec/ilo-management-interface

This blueprint adds support to management Interface for HP ProLiant Servers using iLO client python library.

Problem description

Currently, IloManagement Interface used in IloVirtualMediaIscsiDriver, IloVirtualMediaAgentDriver and PXEAndIloDriver uses IPMIManagement to support management operations like `get_boot_device`, `get_supported_boot_devices`.

This leads to dependency of ipmitool on iLO servers.

Proposed change

- Our proposed change is to do above operations using iLO for consistency, simplicity and correctness.
- Move existing IloManagement class from ilo/deploy.py to ilo/management.py.
- Change the existing IloManagement Class to inherit from base.ManagementInterface instead of ipmitool.IPMIManagement.
- Implement the following methods using iLO client python library.
  - `validate()` - To validate iLO driver specific information. (ilo_username, ilo_password, ilo_address)
  - `get_boot_device()` - To get the current boot device of a node with the indication whether its persistent, or not.

  - `get_supported_boot_devices()` - To get a list of the supported boot devices of a node. The supported boot devices will be `disk`, `pxe` and `cdrom`.

- Move `set_boot_device()` functionality to ManagementInterface and change the current invocations to manager_utils.node_set_boot_device().
- Implementation of `get_sensors_data()` is not in scope of current spec. It is proposed as part of the following blueprint- https://blueprints.launchpad.net/ironic/+spec/send-ilo-health-metrics-to-ceilometer

Alternatives

Continue to use IPMI Interface to management operations.

Data model impact

None

REST API impact

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

The following which are already part of driver_info fields are required:

- `ilo_address` - hostname or IP address of the iLO.
- `ilo_username` - the username for the iLO with administrator privileges.
- `ilo_password` - the password for `ilo_username`.
- `ilo_client_timeout` - the timeout for iLO operations. The default value will be 60 seconds.
- `ilo_client_port` - the port to be used by iLO client for iLO operations. The default value will be 443.

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
anusha-iiitm

Work Items

- Implement `get_boot_device` and `get_supported_boot_devices`.
- Move `set_boot_device()` functionality from ilo/common.py to ilo/management.py.
- Change the current invocations of `set_boot_device` from ilo_common.set_boot_device() to manager_utils.node_set_boot_device().

Dependencies

- Depends on `proliantUtils` library.
- Targeted for HP ProLiant servers with iLO4.

Testing

- Unit tests will be added, mocking proliantutils library.

Upgrades and Backwards Compatibility

None

Documentation Impact

None

References

proliantutils library:

https://github.com/hpproliant/proliantutils

https://pypi.python.org/pypi/proliantutils

5.14.10 Discover node properties and capabilities for iLO drivers

Blueprint URL: https://blueprints.launchpad.net/ironic/+spec/ilo-properties-capabilities-discovery

This proposal adds the ability to introspect/update hardware properties and auto-create ports for HP ProLiant servers via iLO using iLO client python library i.e. proliantutils library as given in reference section.

Problem description

The iLO driver is proposed to be used to discover node properties irrespective of whether OS is deployed on baremetal node or not.

Proposed change

Following mandatory properties will be discovered and updated to node.properties as discussed in http://specs.openstack.org/openstack/ironic-specs/specs/kilo/ironic-node-properties-discovery.html

- memory size
- CPUs
- CPU architecture
- NIC(s) MAC address
- disks

The following additional properties are of interest to iLO drivers and will be discovered and updated to node.properties as capabilities:

- Supported Boot Modes

  capability name : supported_boot_mode possible values : bios, uefi, secure_boot

- iLO Firmware version

  capability name : ilo_firmware_version possible values : it can vary hardware to hardware.

- ROM Firmware version

  capability name : rom_firmware_version possible values : it can vary hardware to hardware.

- Server Name/Model

  capability name : server_model possible values : it can vary hardware to hardware.

- RAID level

  capability name : max_raid_level possible values : 0,1,2,3,4,5,6,10

- secure boot capability

  capability name : support_secure_boot possible values : True, False

- PCI (GPU) devices

  capability name : pci_gpu_devices possible values : count of such devices.

- SR-IOV capabilities

  capability name : sr_iov_devices possible values : count of such devices.

- NIC Capacity

  capability name : nic_capacity possible values : value with unit.

The properties which are already set will be overridden at reinvocation of inspect_hardware(). If a property is not applicable to the hardware or cannot be retrieved from the hardware, the property will not be added/updated in node.properties as capabilities. Even if the property cannot be retrieved from the hardware due to some unknown reasons, the introspection will not return failure as it is same as property not applicable to the hardware.

iLO specific module changes:

- Implement the InspectInterface method inspect_hardware().

Alternatives

These properties can be discovered manually outside the ironic and node.properties updated accordingly with the discovered properties.

Data model impact

None.

Ironic CLI impact

None.

REST API impact

None.

RPC API impact

None.

Driver API impact

None.

Nova driver impact

None.

Security impact

None.

Other end user impact

None.

Scalability impact

None.

Performance Impact

None.

Other deployer impact

None.

Developer impact

None.

Implementation

Assignee(s)

Primary assignee:
agarwalnisha1980

Other Contributors:
wan-yen-hsu

Work Items

- Implementation of the InspectInterface class and its methods inspect_hardware(), validate() and get_properties().

Dependencies

- This feature is targeted for HP ProLiant servers with iLO4 and above. This module might work with older version of iLO (like iLO3), but this will not be officially tested by the iLO driver team.
- Depends on proliantutils library.
- Depends on following also: http://specs.openstack.org/openstack/ironic-specs/specs/kilo/ironic-node-properties-discovery.html

Testing

Unit tests will be added conforming to ironic testing requirements, mocking proliantutils. It will get tested on real hardware by iLO team with the hardware available to the team.

Upgrades and Backwards Compatibility

No impact.

Documentation Impact

None.

References

1. proliantUtils library. (https://github.com/stackforge/proliantutils) (https://pypi.python.org/pypi/proliantutils)

2. Introspect spec. http://specs.openstack.org/openstack/ironic-specs/specs/kilo/ironic-node-properties-discovery.html

5.14.11 Implement Cleaning States

https://blueprints.launchpad.net/ironic/+spec/implement-cleaning-states

When a node has finished a workload, driver interfaces should have the opportunity to run a set of tasks immediately after tear down and before the node is available for scheduling again.

Problem description

- When hardware is recycled from one workload to another, there should be some work done to it to ensure it is ready for another workload. Users will expect each baremetal node to be the same. Applying these baseline tasks will ensure a smoother experience for end users.
- These steps can also be performed (if enabled) on newly enrolled nodes moving from MANAGED to AVAILABLE.
- These baseline tasks should be any task that a) prepares a node for being provisioned to, including making it consistent with other nodes and b) doesnt change the apparent configuration of the machine.
- At a minimum, hard drives should be erased (if enabled)
- Other potentially useful tasks would be resetting BIOS, applying BIOS settings (uniform settings for all nodes, rather than individually on each node), validating firmware integrity and version, verifying hardware matches the node.properties, and booting long running agents [2].
- Some users will require certain security measures be taken before a node can be recycled, such as securely erasing disks, which can be implemented using custom hardware managers in the Ironic Python Agent or with out of band systems that support it.
- The current PXE ramdisk would support a limited subset of cleaning steps, such as erase_devices(). Steps such as verifying the nodes properties will require cooperation of the BMC, Ironic Python Agent, or additions to the current ramdisk.

Proposed change

- The cleaning features will be added behind a config option, to ensure operators have the choice to disable the feature if it is unnecessary for their deployment. For example, some operators may want to disable cleaning on every request and only clean occasionally via ZAPPING.

- Add a decorator *@clean_step(priority)* to decorate steps that should be run as a part of CLEANING. priority is the order in which the step will be run. The function with the highest priority will run first, followed by the one with the second highest priority, etc. If priority is set to 0, the step will not be executed. The argument should be a config option, e.g. *priority=CONF.$interface.$stepname_priority* to give the operator more control over the order steps run in (if at all).
- Add a new function *get_clean_steps()* to the base Interface classes. The base implementation will get a list of functions decorated with *@clean_step*, determine which are enabled, and then return a list of dictionaries representing each step, sorted by priority.
- The return value of *get_clean_steps()* will be a list of dicts with the 3 keys: step, priority and interface, described below:
  - step: function_name,
  - priority: an int or float, used for sorting, described below,
  - interface: interface_name

  Only steps with a priority greater than 0 (enabled steps) will be returned.
- Add a new function *execute_clean_step(clean_step)* to the base Interfaces, which takes one of the dictionaries returned by *get_clean_steps()* as an arg, and execute the specified step.
- Create a new function in the conductor: *clean(task)* to run all enabled clean steps. It will get a list of all enabled steps and execute them by priority. The conductor will track the current step in a new field on the node called clean_step.
- In the event of a tie for priority, the tie breaker will be the interface implementing the function, in the order power, management, deploy interfaces. So if the power and deploy interface both implement a step with priority 10, powers step will be executed first, then the deploy interfaces step.
- If there is a tie for priority within a single interface (an operator inadvertently sets two to the same priority), the conductor will fail to load that interface while starting up, and log errors about the overlapping priorities.
- Using CLEANING, CLEANED, and CLEANFAIL that will be added in the new state machine spec [1]. These states occur between DELETED and AVAILABLE. This will prevent Nova delete commands from taking hours.
- CLEANED will be used much like DELETED: generally as a target provision state. A node will be in CLEANED state after CLEANING completes and until the conductor gets a chance to move it to AVAILABLE.
- Nodes may be put into CLEANING via an API call (described below) only from MANAGED or CLEANFAIL states. MANAGED allows an operator to clean a node before it is available for scheduling. This ensures new nodes are at the same baseline as other, already added nodes.
- The ZAPPING API will allow nodes to go through a single or list of clean_steps from the MANAGED state. These will be operator driven steps via the API, as opposed to the automated CLEANING that occurs after tear_down described in this spec.
- Make the Nova Virt Driver look for CLEANING, CLEANED, and CLEANFAIL states in _wait_for_provision_state() so the node can be removed from a users list of active nodes more quickly. Failures to clean should not be errors for the user and need to be resolved by an operator.

- If a clean fails, the node will be put into CLEANFAIL state, have last_error set appropriately, and be put into maintenance. The node will not be powered off, as a power cycle could damage a node. The operator can then fix the node, and put the nodes target_provision_state back to CLEANED via the API to retry cleaning or skip to AVAILABLE.
- CLEANING will not be performed on rebuilds.

Alternatives

- The interfaces could implement this in tear_down only. It would be slower from a user perspective.
- Most of these actions could be taken during deploy. This would significantly lengthen the amount of time to deploy in some cases (especially with spinning disks)

Data model impact

- node.clean_step will be added as a dict or None field to track which step is currently being performed on the node. This will give the operator more visibility into what a node is doing and allow conductor fail overs during CLEANING to be more simply implemented.
- If a cleaning step needs to store additional information, it should use node.driver_info. For example, the agent interface will store the IPA hardware manager version in driver_info, so it can detect changes and restart cleaning if a new hardware manager is deployed during a cleaning cycle.

REST API impact

- The API will be changed to prevent changing power state or provision state while the node is in a CLEANING state. A node in CLEANFAIL state may be powered on and off via the API, because the operator will likely need to restart the node to fix it.
- The API will allow users to put a node directly into cleaning provision_state with a POST, the same as how provision state is changed anywhere else in Ironic. This can be useful for verifying newly added nodes or if an operator wants to put a fleet of inactive servers into a known state. A node can only be put into CLEANING state from MANAGED or CLEANFAIL states.
- Nodes in CLEANFAIL may be put into CLEANING or AVAILABLE state, as determined by the operator.
- An API endpoint should be added to allow operators to see currently enabled clean steps and their ordering. This will be a GET endpoint at /nodes/<uuid>/cleaning/steps and will return the exact data noted above for *get_clean_steps()*, as a JSON document and ordered by priority.
- GET requests to the nodes API (/nodes/<uuid>) and node detail API (/nodes/details) should return the current node.clean_step as well.

RPC API impact

Cleaning of a node will need to be available via RPC, so the API servers can put a node into CLEANING from MANAGED or CLEANFAIL states.

At the end of a tear down, the conductor will RPC call() the do_node_clean() method of the conductor.

As the states will first be added as no-ops in the new state machine spec, upgrading wont be a problem.

Driver API impact

- The BaseDriver will have a *get_clean_steps()* and *execute_clean_steps()* functions added and implemented.

  def get_clean_steps(task):
  Return the clean steps this interface can perform on a node

  param task
  a task from TaskManager.

  returns
  a list of dictionaries as noted above

  def execute_clean_steps(task, step):
  Execute the given clean step on the task.node

  param task
  a task from TaskManager.

  param step
  a step from get_clean_steps()

  raises CleanStepFailed
  if the step fails

- Testing will be similar to other driver interfaces and each interface will be expected to test their implementation thoroughly.
- Existing interfaces can choose to not implement the new API with no effect, as they will be added in the base classes.

Nova driver impact

- Nova driver will look for the clean states when determining if unprovisioning succeeded or not.
- If Nova is upgraded first, nothing will change. The driver will continue to be in the tear_down state until the node goes to AVAILABLE.

Security impact

- Security will be improved by adding erasing of disks [3].
- It should be noted in documentation that there are still attack vectors if baremetal nodes are given to untrusted users or if a baremetal node is compromised.
- If the API is called to set a node into a clean state, that node could be tied up for potentially hours. If run against enough nodes in a cluster by a bad actor, the cluster could run out of capacity quickly. These APIs by default require admin privileges. However, a user could provision and unprovision nodes quickly, leading to a denial of service. Quotas could mitigate this issue.

Other end user impact

None

Scalability impact

None

Performance Impact

- There will be additional calls to the hardware to perform the cleaning steps. The steps could take hours, which will mean the time to recycle could be much higher than before.
- The node will be locked for the duration of the clean.
- Database calls will increase, because the state is saved after each cleaning step that requires a reboot or long running process, as well as saving the current clean_step before it begins execution of the step.
- Rebalances, in the worst case, will require the node to redo one step based on the cleaning_step. This could happen if a conductor dies while it owns a node that is doing a long running process. clean_steps should be implemented as idempotent actions, to avoid issues here.

Other deployer impact

- Deployers will need to inspect which clean steps are being performed and adjust whether each step is performed and at what priority if the defaults dont work for their environment.
- If Ironic is updated first, nodes that are torn down may take additional time and will likely time out in unprovision. This would only happen if Ironic was updated before Nova, and a interface that implements clean which takes a large amount of time was enabled and used. This will need to be documented.

Developer impact

- Drivers will need to call any functions they deem necessary to clean a node, and possibly implement those functions. They may add config options to enable or disable those features.

Implementation

Assignee(s)

Primary assignee:
JoshNang

Other contributors:
jroll JayF

Work Items

- Add clean() to the conductor
- Add get_clean_steps() and execute_clean_step() to the BaseDriver interface.
- Add @clean_step() decorator
- Add API checks for clean states and allow CLEANED as a provision target state
- Add API end point /nodes/<uuid>/cleaning/steps
- Add support for erase_disks in PXE interface

- Add cleaning support to IPA
- Add Nova driver support

Dependencies

- Ironic State Machine: https://review.openstack.org/#/c/133828/. Both are attempting to add CLEANING/CLEANED/CLEANFAIL. If this is implemented without a new clean state, users will see a node in deleting state in Nova for potentially hours, eating up quota.
- Not required, but would be helpful: External event callback API would be helpful for the Agent deploy interface (and probably others) implementation of clean: https://review.openstack.org/#/c/99770/.

Testing

- Tempest will have to be adapted to support running a clean as part of its normal provision/unprovision tests.
- Drivers implementing cleaning will be expected to test their added features.

Upgrades and Backwards Compatibility

- The changes to the REST API to allow a node to go from MANAGED or CLEANFAIL to CLEANED will require the user to specify the new state: CLEANED. Therefore, it shouldnt break backwards compatibility. The only change existing users/tools may see is an extended period where nodes are unable to be powered off via the API.

Documentation Impact

- There should be very clear documentation about how cleaning works, how the steps are ordered, what they do, and how operators can enable, disable, and reprioritize them. This is essential for operators to understand if they are going to use cleaning. The differences in between interfaces for cleaning will also need to be spelled out.
- The Ironic driver interface changes, the Nova driver support and changes to Ironic API will need to be documented.
- We should document the security problems that still exist, even with cleaning enabled.

References

1: http://specs.openstack.org/openstack/ironic-specs/specs/kilo/new-ironic-state-machine.html

2: https://review.openstack.org/#/c/102405/

3: https://bugs.launchpad.net/ironic/+bug/1174153

5.14.12 In-band hardware properties introspection via ironic-discoverd

https://blueprints.launchpad.net/ironic/+spec/inband-properties-discovery

This spec adds support for introspecting hardware properties using *ironic-discoverd* project for drivers not supporting out-of-band introspection.

ironic-discoverd[1] is a StackForge project for conducting hardware properties inspection via booting a special discovery ramdisk[2] and interrogating hardware from within it.

Note

Term hardware discovery is used through this spec as a synonym to hardware introspection or hardware inspection as opposed to new hardware auto-discovery, which is not covered by this spec.

Problem description

Currently there is no generic way in Ironic to inspect scheduling properties of a given piece of hardware. While multiple out-of-band methods are proposed, they are restricted each to a particular vendor (currently HP and DELL). This proposal will make hardware introspection possible for every driver that can power on and off the machine.

A couple of future use cases may need in-band discovery even on hardware supporting out-of-band discovery, namely:

- Auto-discovery of nodes. While out of scope for now, it may be considered later, and may require in-band discovery, depending on whether particular vendor support it out-of-band.
- Vendor-specific extensions for the introspection, discovering properties that are not available via out-of-band means etc.

Note

This spec does not touch any of these use cases directly.

Proposed change

This spec proposes implementing `InspectInterface` defined in the parent spec[3] via using *ironic-discoverd*.

Note

ironic-discoverd 1.0.0 (to be released in the beginning of February) will be a requirement, so e.g. some client functions might be missing from the current stable 0.2 release documentation. See *Dependencies* for details.

- Add `[discoverd]` configuration section:
  - Add configuration option `service_url` (string, no default). This is a URL of *ironic-discoverd* which will be contacted by Ironic for starting the introspection process. E.g. `http://127.0.0.1:5050/v1`.

    Note

    Port can be changed via *ironic-discoverd* configuration file.

[1] *ironic-discoverd*: https://pypi.python.org/pypi/ironic-discoverd
[2] Reference ramdisk: http://bit.ly/1yD9nnq
[3] Parent spec: https://review.openstack.org/#/c/100951/

  - Add configuration option `enabled` (boolean, default `False`), that will enable or disable the discoverd introspection specifically. By default it will be disabled and `DiscoverdInspect.inspect_hardware` will raise new `DiscoverdDisabled` exception with the explanation.

    Note

    This option does not affect other inspection implementation.

- Create `DiscoverdInspect` class implementing `InspectInterface` in the following way:
  - During driver initialization the constructor (__init__) verifies that *inspection_service_url* is provided and `ironic_discoverd.client` module can be imported, otherwise fail. It will also check that we have the required version of *ironic-discoverd* - see *Dependencies*. These checks obviously will be skipped, if *ironic-discoverd* support is disabled via configuration - see above.
  - `inspect_hardware` releases a task manager lock on a node (so that *ironic-discoverd* can operate on it) and calls to `ironic_discoverd.client.introspect` providing node UUID.

    Note

    Without releasing lock that was acquired by generic inspection code for creating a *task*, *ironic-discoverd* wont be able to manipulate a node.

  - *ironic-discoverd* will update scheduling data after successful introspection.
  - `DiscoverdInspect` will also poll state endpoint[4] of *ironic-discoverd* via client API `ironic_discoverd.client.get_status`. It will use a driver-specific periodic task as suggested in[5].

    On success node will be advanced to the next state. On error `Node.last_error` will be updated and node will be moved to INSPECTFAIL state.

    Note

    If we decide not to implement driver-specific periodic tasks for Kilo cycle, a LoopingCall at the end of inspect_hardware() method will be used instead.

- Add `DiscoverdInspect` as `InspectInterface` implementation for `pxe_ipmitool`, `pxe_ipminative` and `pxe_ssh` drivers (the latter for easier testing).

High-level schema of how inspection will work:

- `DiscoverdInspect` calls to *ironic-discoverd* API
- *ironic-discoverd* communicates with Ironic via regular API for checking the current node state.
- *ironic-discoverd* asks Ironic to reboot the machine using the power interface.
- The machine is PXE-booted using PXE server (usually *dnsmasq*) installed along with *ironic-discoverd*.

[4] *ironic-discoverd* status API blueprint: https://blueprints.launchpad.net/ironic-discoverd/+spec/get-status-api
[5] Driver-specific periodic tasks spec: https://review.openstack.org/#/c/135589

Note

It was suggested that *ironic-discoverd* use out-of-band boot methods like virtual media, if supported by the driver. While an interesting feature to consider, its not in the near-term plans for *ironic-discoverd* and thus is not covered by this spec.

Note

ironic-discoverd does not use Neutron, as Neutron does not provide means apply specific DHCP options to all **unknown** machines (i.e. with MACs that do not have ports). If one day it does provide such functionality, *ironic-discoverd* will switch to it and stop managing *dnsmasq* directly. Please refer to the README[Page 736, 1] for details.

Note

ironic-discoverd avoids conflicts with Neutron by managing firewall access to PXE port. Only MACs that are not known to Ironic are allowed to PXE-boot via *dnsmasq* instance managed by *ironic-discoverd*.

- The ramdisk calls back to *ironic-discoverd*.
- *ironic-discoverd* communication with Ironic via regular API to update node state and powers off the node

Alternatives

- We could stay with out-of-band discovery only. As stated above, its not covering all hardware.
- In-band discovery could be implemented within Ironic itself, without 3rd party service. This is believed to be a unnecessary complication to Ironic code base.

Data model impact

No direct impact expected. *ironic-discoverd* will set scheduling properties in `Node.properties` field.

REST API impact

No direct REST API impact.

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Security impact

The code within Ironic has no security impact. Presence of *ironic-discoverd* itself has one security issue:

- Endpoint receiving data from the ramdisk is not authenticated. For this reason *ironic-discoverd* will verify that node is in its internal cache before updating it.

  Due to this, the current policy is to never overwrite existing properties, only set the missing ones. It will be possible to alter the behavior via *ironic-discoverd* configuration file. Please refer to the parent spec[Page 736, 3] for discussion.

Other end user impact

Operators will be able to gather properties from hardware which does not support out-of-band introspection via vendor-specific drivers.

Scalability impact

- *ironic-discoverd* currently requires one PXE boot server and one TFTP server to serve all the requests to boot discovery ramdisk. This is the only thing that seriously limits the scalability of *ironic-discoverd*.

  In the future well be looking into how to make *ironic-discoverd* work in the redundant setup, but currently its not supported.

- *ironic-discoverd* also may require more network calls than out-of-band inspection.

I believe that these concerns are not critical, if discovery happens not too often and in reasonable bulks of nodes.

Performance Impact

Call to *ironic-discoverd* is mostly async, only basic sanity checks are done in a sync fashion, before returning control back to the conductor.

Other deployer impact

- New option `discoverd.enabled` (boolean, default `False`) - whether to enable inspection via *ironic-discoverd*
- New option: `discoverd.service_url` (string, no default) with the URL of *ironic-discoverd*
- *ironic-discoverd* and required services (like dnsmasq and TFTP server) should be deployed and managed separately, see the README[Page 736, 1] for details.

  ironic-discoverd will ensure that these services wont interfere with existing Neutron installation managing DHCP for the nodes.

  A special ramdisk[Page 736, 2] should be built (e.g. using *diskimage-builder*) and located in TFTP root directory - again see README[Page 736, 1].

- *ironic-discoverd* support will be optional and disabled by default, thus no impact on fresh or upgraded installations.

Developer impact

Driver developers may use `DiscoverdInspect` to provide in-band hardware discovery for their drivers.

Implementation

Assignee(s)

Primary assignee:
Dmitry Tantsur, LP: divius, IRC: dtantsur

Work Items

- Add new configuration options.
- Implement `DiscoverdInspect` interface.

Dependencies

- Generic bits for the discovery[Page 736, 3]
- *ironic-discoverd* package should be installed from PyPI or RDO to enable this feature. Version 1.0.0[6] (to be released in the beginning of February) will be a requirement for the new interface. Drivers using it will refuse to load without it, if `discoverd.enabled` is set to true.
- Driver-specific periodic task as specified in[Page 737, 5] is suggested for use. If that spec is not accepted, well fall back to using a looping call instead.

Testing

- Unit testing with mocking `ironic_discoverd.client` for now
- As a follow-up I hope to add support for *ironic-discoverd* to devstack and then have a functional test. This however will consume much time and depends on the functional testing discussions going on.

  As stated above, *ironic-discoverd* will be disabled by default.

Upgrades and Backwards Compatibility

None

Documentation Impact

It should be documented how to enable in-band inspection within Ironic. The documentation should point to *ironic-discoverd* README for the installation instruction.

References

5.14.13 iRMC Management Driver for Ironic

https://blueprints.launchpad.net/ironic/+spec/irmc-management-driver

The proposal presents the work required to add support for management standard interface for FUJITSU PRIMERGY iRMC, integrated Remote Management Controller, Drivers in Ironic.

[6] *ironic-discoverd* 1.0.0 release status: https://bugs.launchpad.net/ironic-discoverd/+milestone/1.0.0

Problem description

FUJITSU PRIMERGY iRMC allows IPMI to get/set boot mode either legacy BIOS or UEFI, and SCCI to get sensor data. However the current Ironic standard Management Module, ipmitool.IPMIManagement, cannot make use of the iRMCs capabilities.

Proposed change

This module inherits ipmitool.IPMIManagement and overrides the following two functions for iRMC Drivers, namely pxe_irmc, iscsi_irmc, and agent_irmc[1], in order to make use of iRMC IPMI boot mode get/set capability and SCCI get sensor data.

This module will be re-factored, when Ironic set/get boot mode I/F is standardized in Ironic Management Interface.

- set_boot_device() - If boot_mode:uefi is specified in capabilities parameter within properties field of an Ironic node, this function issues IPMI Set System Boot Options Command with setting on the bit 5 of data1, BIOS Boot Type to UEFI, in the parameter selector 5 to iRMC. Otherwise this function just calls the parent class function, ipmitool.IPMIManagement.set_boot_device() as default.
- get_sensors_data() - If optional parameter sensor_method=scci is specified in [irmc] section of the ironic configuration file, this function gets sensors data via iRMC SCCI which returns not only standard but also vendor specific sensor data. iRMC SCCI uses python-scciclient package. Otherwise, if the optional parameter is the default value sensor_method=ipmitool, this function just calls the parent class function, ipmitool.IPMIManagement.get_sensors_data() as default.

Alternatives

There is no alternative if bare metal node is necessary for booting in UEFI mode automatically.

IPMI management module can be used only if deployer sets boot mode of a bare metal node manually into UEFI.

Data model impact

None

REST API impact

None

[1] Driver consists of five elements. In the initial implementation, iRMC driver supports three combinations out of all combinations.

driver	power	boot	deploy	management	console
pxe_irmc	irmc	pxe	iscsi	irmc	ipmi
iscsi_irmc	irmc	irmc	iscsi	irmc	ipmi
agent_irmc	irmc	irmc	agent	irmc	ipmi

Other combinations are considered in the next development cycle based on customers feedback.

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Security impact

Admin credentials will be stored unencrypted in the DB and they will be visible in the driver_info field of the node when a node-show is issued. But only the ironic admin user will have access to the Ironic DB and node details.

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

- In order to use iRMC driver, iRMC S4 and beyond is required. Deployer is notified by error message if the iRMC version is not valid.
- The driver_info fields and the [irmc] section parameters in the ironic configuration file are necessary which are specified in iRMC Power Driver for Ironic.
- Boot mode maybe set to BIOS or UEFI following command described in Bare Metal Service Installation Guide: Boot mode support or iLO drivers: Boot mode support.
- To configure a node in BIOS mode:

```
ironic node-update <node-uuid> add properties/capabilities='boot_mode:bios
↪'
```

- To configure a node in UEFI mode:

```
ironic node-update <node-uuid> add properties/capabilities='boot_mode:uefi
↪'
```

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
Naohiro Tamura (naohirot)

Other contributors:
None

Work Items

- Implement iRMC Management Module for the iRMC Drivers (pxe_irmc, iscsi_irmc, agent_irmc) by inheriting ipmitool.IPMIManagement and overrides set_boot_device() and get_sensors_data().

Dependencies

- This feature requires iRMC S4 and beyond that is at least BX S4 or RX S8 generation of FUJITSU PRIMERGY servers.
- This feature requires ipmitool command and python-scciclient package.
- This feature implemented based on the iRMC Drivers (pxe_irmc, iscsi_irmc, agent_irmc) which are defined in iRMC Power Driver for Ironic and iRMC Virtual Media Deploy Driver for Ironic.

Testing

- Unit Tests
- Fujitsu plans Third-party CI Tests

Upgrades and Backwards Compatibility

The default behavior of this driver remains compatible with ipmitool.IPMIManagement.

Documentation Impact

The required driver_info fields and [irmc] section parameters in the ironic configuration file need be included in the documentation to instruct operators how to use Ironic with iRMC.

References

- FUJITSU Software ServerView Suite, Remote Management, iRMC S4 - integrated Remote Management Controller
- iRMC Power Driver for Ironic
- iRMC Virtual Media Deploy Driver for Ironic
- python-scciclient package
- New driver ManagementInterface
- DRAC Management driver for Ironic

- iLO Management Interface
- iLO drivers: Boot mode support
- Bare Metal Service Installation Guide: Boot mode support

5.14.14 iRMC Power Driver for Ironic

https://blueprints.launchpad.net/ironic/+spec/irmc-power-driver

The proposal presents the work required to add support for power management features for FUJITSU PRIMERGY iRMC, integrated Remote Management Controller, Drivers in Ironic.

Problem description

FUJITSU PRIMERGY iRMC is a BMC from FUJITSU offering a remote system management. This proposal adds the power management capabilities for iRMC.

Proposed change

Adding new iRMC Driver, namely pxe_irmc, to the list of available drivers in Ironic and implementing the iRMC power module to interact with ServerView Common Command Interface (SCCI) described in FUJITSU Software ServerView Suite, Remote Management, iRMC S4 - integrated Remote Management Controller

iRMC supports WS-MAN, CIM, SMASH CLP, IPMI, SNMP, and etc. ServerView Common Command Interface (SCCI), however, is chosen since it is the most capable among them.

ServerView Common Command Interface (SCCI) uses python-scciclient package.

Alternatives

Standard IPMI can be used for the power management.

Data model impact

None

REST API impact

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Security impact

Admin credentials will be stored unencrypted in the DB and they will be visible in the driver_info field of the node when a node-show is issued. But only the ironic admin user will have access to the Ironic DB and node details.

Other end user impact

The following driver_info fields can be provided:

- irmc_address: hostname or IP of iRMC
- irmc_username: username for iRMC with administrator privileges
- irmc_password: password for irmc_username
- irmc_port: port number of iRMC (optional)
- irmc_auth_method: authentication method for iRMC (optional)

The following parameters are added into newly created [irmc] section in the ironic configuration file which is typically located at /etc/ironic/ironic.conf.

- port: default value of iRMC (80 or 443) port number. The default value is 443.
- auth_method: default value of iRMC authentication method (basic or digest). The default value is basic.
- client_timeout: default timeout for SCCI operations. The default value is 60 seconds.

Scalability impact

None

Performance Impact

None

Other deployer impact

None

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
Naohiro Tamura (naohirot)

Other contributors:
None

Work Items

- Add iRMC Driver (pxe_irmc)
- Implement iRMC power module for the iRMC Drivers

Dependencies

- This feature requires at least BX S4 or RX S8 generation of FUJITSU PRIMERGY servers.
- This feature requires python-scciclient library. This dependency will be checked on calling __init__() of iRMC driver.

Testing

- Unit Tests with mocking python-scciclient library.
- Fujitsu plans Third-party CI Tests

Upgrades and Backwards Compatibility

None

Documentation Impact

The required driver_info fields and [irmc] section parameters in the ironic configuration file need be included in the documentation to instruct operators how to use Ironic with iRMC.

References

- FUJITSU Software ServerView Suite, Remote Management, iRMC S4 - integrated Remote Management Controller
- iRMC Virtual Media Deploy Driver for Ironic
- iRMC Management Driver for Ironic
- python-scciclient package
- DRAC Power Driver for Ironic
- iLO Power Driver for Ironic

5.14.15 Discover node properties with node-set-provision-state

Blueprint URL: https://blueprints.launchpad.net/ironic/+spec/ironic-node-properties-discovery

This proposal adds the ability to perform out-of-band node inspection and automatically update node properties using node-set-provision-state (new target inspect) process. The same set of APIs can be used by in-band properties discovery too.

Problem description

Today, Ironic is unable to automatically detect node properties which are required for scheduling or for deploy (ports creation).

The properties required by scheduler/deploy:

- memory size

- Disk Size
- CPU count
- CPU Arch
- NIC(s) MAC address

Proposed change

The spec proposes to add these abilities:

1. Discover the hardware properties and update the node.properties. It update all the properties with the new set values, even if the values were already set.
2. Create ports for the NIC MAC addresses discovered. This will be done automatically after the NICS are discovered with above step. If a port already exists, it will not create a new port for that MAC address. It will take care of adding as well as deleting of the ports for the NIC changes. The ports no longer associated with the node at the time of discovery will be auto deleted. Ideally the ports shall be created for PXE enabled mac addresses, however it is each driver responsibility to get list of pxe enabled mac addresses from the hardware.

Following would be done to achieve properties inspection:

- A new target inspect would be added to the end point provision /v1/nodes/<uuid>/states/provision
- When inspection is finished, puts node.provision_state as MANAGED and node.target_provision_state as None.
- The possible new states are: INSPECTING, INSPECTFAIL, MANAGED, INSPECTED.

  Refer to [1] for more details on each state.

  Initial state : node.provision_state = INSPECTING node.target_provision_state = INSPECTED

  On Success: Final State : node.provision_state = MANAGED node.target_provision_state = None

  On Failure: Final State: node.provision_state = INSPECTFAIL node.target_provision_state = None And the error is updated at the last error.

  The provision state will be marked as INSPECTFAIL if any of the properties (scheduler required properties) could not be inspected. In this case, no properties will be updated. So its either all or none.

  A node must be in MANAGED state before initiating inspection.
- The properties values will be overwritten with every invocation of discovery APIs. It is done so because there may be addition/deletions in the hardware between two invocations of the discovery CLI/APIs.
- Implement timeouts for inspection. There should be a periodic task in the conductor, checking that every mapped node is not in INSPECTING state for more then `inspect_timeout`. This will be added as a config option in /etc/ironic/ironic.conf.

The Node.properties introspected properties will be updated whenever introspection is executed.

Hardware properties to be discovered:

The following properties would be discovered :

- NICs mac addresses
- cpus
- cpu_arch
- local_gb
- memory_mb

The NICs data would be used to auto-create the ports. The other properties are mandatory properties for the scheduler, hence will be updated in Node.properties field.

Alternatives

The discovery can be initiated by node-create with single request to the API as below:

POST /v1/nodes/?discover=true :

In this, it requires the Nodes.post to be completely asynchronous. This would be a compatibility break for node-create as it requires a node object to be returned back. On the other hand, if discovery is done synchronously then Nodes.post will become slow. In this if Nodes.post still returns the node object without properties updated to the CLI and still do discovery in the background then Nodes.post becomes half synchronous and half asynchronous, then it will be wrong for REST API structure as the REST API shall be either synchronous or asynchronous. Hence, here node-create becomes partly synchronous and partly asynchronous. Given above reasons, this approach is not chosen.

The auto-discovery can also be used for discovering node properties but it also doesnt provide the ability to discover properties at node-create and node-set-provision-state.

Data model impact

The discovered properties will be stored as part of the properties field of the node. It will add two fields to the *nodes* table:

1. *last_inspected*: It will be updated with time (when last inspection finished).
2. *inspection_started_at*: It will store the start time of invocation of inspection. This will be required to calculate the *inspection_timeout*. It shall be cleared out after inspection finishes or timeout.

Ironic CLI impact

The node-set-provision-state target inspect need to be defined. The synopsis will look like:

usage: ironic node-set-provision-state <UUID> inspect

REST API impact

The endpoint provision will be enhanced with a new target:

- PUT /v1/nodes/<uuid>/states/provision {target: inspect}
  - Changes provision_state as INSPECTING.
  - Changes target_provision_state as INSPECTED.

  - Method type: PUT
  - Normal response code: 202
  - Expected errors:
    * 404 if the node with <uuid> does not exist.
    * 400 if a conductor for the nodes driver cannot be found.
    * 409 CONFLICT, if the node cannot be transitioned to INSPECTING state.
  - URL: /v1/nodes/<uuid>/states/provision {target: inspect}
  - URL parameters: None.
  - Response body is empty if successful.
  - When inspection is finished, puts node.provision_state as MANAGED and node.target_provision_state as None.

RPC API impact

- A new rpcapi method inspect_hardware() will be added. It will be synchronous call to the conductor and will spawn a worker thread to perform discovery. This will allow the API service to receive the acknowledgement from the conductor that the inspecting has been initiated and returns status 202 to the client.

Driver API impact

It will add new interface InspectInterface with the method inspect_hardware():

```
def inspect_hardware(self, task):
    """Inspect hardware.

    :param task: a task from TaskManager.

    """
```

A driver may choose to implement the InspectInterface.

Since InspectInterface is a standard interface, following methods will also be added:

- validate()
- get_properties()

Nova driver impact

None.

Security impact

None.

Other end user impact

This feature will improve user experience as users no longer need to manually update the node properties info.

Scalability impact

None.

Performance Impact

None.

Other deployer impact

The `inspect_timeout` is introduced in the ironic.conf under conductor. The default value for same shall be 1800 secs as required by in-band implementations.

Developer impact

The drivers who need to implement base.InspectInterface(), may decide to implement/define the abstract methods added by this proposal.

Implementation

Assignee(s)

Primary assignee:
Nisha Agarwal (launchpad ID: agarwalnisha1980, IRC login ID: Nisha)

Other Contributors:
Wan-Yen Hsu (launchpad ID : wan-yen-hsu, IRC login ID: wanyen)

Work Items

- A new rpcapi method inspect_hardware() is added which will invoke the InspectInterface for discovering, updating node properties and creating/updating ports.
- Add a new interface as InspectInterface to ironic driver.
- Adding new method inspect_hardware to class InspectInterface.
- Add new elements `last_inspected` and `inspection_started_at` to the *nodes* table. Ironic cli will be changed to show these fields while running `ironic node-show`.
- The node.last_inspected will be updated with the last discovered time.
- The node.inspection_started_at will be updated with the time when inspection was initiated. This will help to check the timeout for inspection. The periodic task needs to be created for the same.
- The reference implementation will be done for iLO drivers.
- Add a new target inspect to node-set-provision-state for updating the hardware properties for already registered node.(ironic-client changes)

Dependencies

Requires implementation of http://specs.openstack.org/openstack/ironic-specs/specs/kilo/new-ironic-state-machine.html for the states MANAGED, INSPECTED, INSPECTING and INSPECTFAIL.

Testing

Unit tests will be added conforming to ironic testing requirements. The test suites for tempest can be written for specific implementations.

Upgrades and Backwards Compatibility

No impact.

Documentation Impact

It needs to be documented properly.

References

[1] All possible states for a ironic node spec: http://specs.openstack.org/openstack/ironic-specs/specs/kilo/new-ironic-state-machine.html

5.14.16 Add support for VirtualBox through WebService

https://blueprints.launchpad.net/ironic/+spec/ironic-virtualbox-webservice-support

This change proposes to add `PowerInterface` and `ManagementInterface` of VirtualBox for testing purposes through VirtualBox WebService.

Problem description

Some developers run Windows as the operating system on their development machines. Such developers may use a linux VM running in VirtualBox as their cloud controller. There is no way for the developers to do testing of Ironic on their laptop using VMs in the same VirtualBox Windows host (with hardware-assisted virtualization) as bare metal nodes. Developers may choose to run kvm/qemu inside their VirtualBox VM but nested virtualization is really slow (as there is no hardware assistance).

Currently, Ironic has support for using a VirtualBox VM as a bare metal target and do provisioning on it. It works by connecting via SSH into the VirtualBox host and running commands using VBoxManage. This works well if you have VirtualBox installed on a Linux box. But when VirtualBox is installed on a Windows box, configuring and getting SSH to work with VBoxManage is a difficult (if not impossible) due to following reasons:

- Windows doesnt come with native SSH support and one needs to use some third-party software to enable SSH support on Windows.
- Even after configuring SSH, VBoxManage doesnt work remotely due to how Windows manages user accounts - the native Windows user account is different from the corresponding SSH user account, and VBoxManage doesnt work properly when done with SSH user account.
- Even after tweaking policies of VirtualBox application, the remote VBoxManage and VBoxSvc dont sync each other properly and often results in a crash.

Proposed change

- VirtualBox comes with a very friendly WebService to manage the VMs remotely. This works by talking to a WebService running on the VirtualBox host using SOAP.
- A new python library named `pyremotevbox` will be written and will be available separately in GitHub and PyPI. Currently it is hosted in GitHub.
- Write a new implementation of `PowerInterface` and `ManagementInterface` named `VirtualBoxPower` and `VirtualBoxManagement` which uses the new python library to manage VirtualBox VMs.
- Create new drivers `pxe_vbox`, `agent_vbox` for deploying on Virtualbox VMs. Also create a `fake_vbox` driver for testing purposes with fake deploy.
- This can also be used by users running VirtualBox on other operating systems where it is supported (other than Windows). They may also use SSHPower and SSHManagement if VirtualBox is running on linux machines. The advantage of this module over ssh ones is that, it will be faster (because ssh modules first ssh to system and then run VBoxManage command which does the same thing, so time for doing ssh is extra). The disadvantage is that VirtualBox webservice should be running all the time (which is not required for ssh ones).

Alternatives

Developers using Windows can continue to use nested virtualization but it is really slow. Also getting SSH to work with Windows for VBoxManage is very difficult and buggy.

Data model impact

None.

REST API impact

None.

RPC API impact

None.

Driver API impact

None.

Nova driver impact

None.

Security impact

None. This is used only on developers own systems for testing purposes.

Other end user impact

None.

Scalability impact

None.

Performance Impact

Developers running Windows will find it very fast to deploy on VMs run with hardware-assisted virtualization rather than nested virtualization.

Other deployer impact

The following driver_info fields are required:

- `vbox_address` - hostname or IP address of the VirtualBox host.
- `vbox_username` - the username for the VirtualBox host.
- `vbox_password` - the password for `vbox_username`
- `vbox_port` - the port to be used by VirtualBox Web Service. The default value will be 18083.
- `vbox_vmname` - the name of the VM in VirtualBox acting as bare metal.

Developer impact

None.

Implementation

Assignee(s)

rameshg87

Work Items

- Add VirtualBoxPowerInterface.
- Add VirtualBoxManagementInterface

Dependencies

- Depends on pyremotevbox library which is being developed. This library will be available in GitHub and PyPI for developers to install on their laptop and will have Apache license.

Testing

Unit tests will be added.

Upgrades and Backwards Compatibility

None.

Documentation Impact

How to use the changes with Windows VirtualBox will be documented in Ironic wiki.

References

None

5.14.17 Local boot support with partition images

https://blueprints.launchpad.net/ironic/+spec/local-boot-support-with-partition-images

This blueprint proposes to accept the boot option specified by Nova flavor and make the subsequent reboot of the baremetal from local hard disk drive instead of pxe or vmedia, depending on the boot option selected by the user.

Problem description

At present Ironic drivers that deploy partition images require Ironic-conductor for subsequent reboots of the node. The subsequent reboot of the node will happen either through pxe or using virtual media. Also there is no way a Nova user can specify the boot option, local/netboot for deploy using partition image.

Proposed change

- Nova Ironic driver should read the specified boot option `capabilities:boot_option` flavor key which should be passed through *node.instance_info* field by Nova Ironic driver.
- Ironic will then pass this information to the deploy ramdisk via kernel cmdline in the PXE template, set the boot device to HDD persistently and clean up the PXE configuration files after the deployment is completed.
- The deploy ramdisk to check the parameter in the kernel cmdline and handle the bootloader installation.

Note

For setting the local boot the images being deployed should have grub2 installed.

Windows images wont be supported as part of this spec.

Creating an EFI boot partition, including the EFI modules and managing the bootloader variables via efibootmgr wont be supported as part of this spec.

Alternatives

- Use localboot element from Disk image builder:

  While generating the partition images using disk-image-builder, we can use localboot element, which is available in tripleo-image-elements project. The downside of using this is that the local boot will be enabled during the first boot after node deploy. So it requires two resets of the server to enable localboot. Also the Ironic-conductor is not aware of this change and continue to provide the pxe or vmedia boot for the reboot operation done from conductor.

- Use whole disk image to achieve local boot:

  The other way to achieve a local boot is to use whole disk image for deploy. Right now, this can be achieved by using agent driver only.

Data model impact

None.

REST API impact

None.

RPC API impact

None.

Driver API impact

None.

Nova driver impact

Nova Ironic driver need to pass down the `capabilities:boot_option` from Nova flavor to Ironic *node.instance_info* field.

Security impact

Local boot is a double-edged problem. On one hand, in case of a electricity outage the customer nodes that are configured to local boot could potentially boot up before the control plane. On the other hand if electricity outage causes the control plane to not boot after is also a problem. So, this spec makes local boot and net boot configurable per instance, deployers should be aware of that when deploying their clouds.

Other end user impact

- Deployer can specify the boot option request in Nova flavor as capability. `capabilities:boot_option=local` or `capabilities:boot_option=netboot` (default).
- Set `boot_option:local` or `netboot` as capability in node.properties.

Scalability impact

This can improve scalability by the fact that there will be less network traffic not having to transfer a kernel and ramdisk over the network to boot a node.

Performance Impact

None.

Other deployer impact

- The image being deployed should have grub2 installed.

Developer impact

None.

Implementation

Assignee(s)

Primary assignee:
lucasagomes

Other contributors:
faizan-barmawer

Work Items

- Implement the code in Ironic that will check for the *boot_option* parameter in the *node.instance_info* field. If set to local we have to:
  - Pass the information down to the deploy ramdisk via the kernel cmdline in the PXE configuration file.
  - Delete the PXE configuration files for that node after the deployment is completed.
  - Set the boot device to HDD persistently before the final reboot.
- Implement the code in the deploy ramdisk that will look at the parameter passed via the kernel cmdline and install the bootloader on the disk as part of the deployment.

Dependencies

- Require this Nova virt Ironic driver fix to pass down the capabilities from Nova flavor to Ironic nodes instance info field. See https://review.openstack.org/141012

Testing

- Unit testing.

Upgrades and Backwards Compatibility

None.

Documentation Impact

- Make changes to Ironic install guide.

References

- PoC patches:
  - Nova: https://review.openstack.org/146619
  - Ironic: https://review.openstack.org/146189

- DIB: https://review.openstack.org/146097

5.14.18 Ironic Logical Names

Include the URL of your launchpad blueprint:

https://blueprints.launchpad.net/ironic/+spec/logical-names

Everything that is tracked in Ironic is done via UUID. This isnt very friendly to humans. We should support nodes being referenced by a logical name, in addition to the current method of being able to refer to them via UUID. This should be supported in the REST API and in our command-line client.

Problem description

Operators, and other humans, that use Ironic find it awkward and error-prone to use UUIDs to refer to entities that are tracked by Ironic.

However computers, and extremely geekly people, prefer to track things by the canonical identifier - the UUID.

While humans are more likely to use the command-line tools, computers are more likely to use the REST API. However the opposite is also true, consequently both the API and command-line client require updating to support logical names for nodes.

Its useful to be able to assign semantic meaning to nodes - for example datacenter location (i.e. DC6), or node function (i.e. database) - to assist operators in managing nodes in their network. The semantic identifier of a node is analogous to the hostname for the node, and may indeed correlate.

An example of this might be where the logical name DC6-db-17 is associated with a node with UUID 9e592cbe-e492-4e4f-bf8f-4c9e0ad1868f. In all interactions with ironic, the nodes UUID or logical name can be used to identify the specific node.

Proposed change

We propose adding a new concept to ironic, that being the <logical name>, which can be used interchangeably with the <node uuid>. Everywhere a <node uuid> can be specified, we should be able to instead specify a <logical name>, if such an association exists for that node. This should be the case for the REST API and python-ironicclient.

At the REST API level, the mechanisms to set/retrieve node fields will be used to set/retrieve the logical name for a node.

At the python-ironiclient level, support will be added to manage associations between a node and a <logical-name>. See further details below.

Where a change in the interface is required, such as a new attribute on an object, a new key in a dictionary, or a new field in POST data, the <logical name> will be referred to as name (unless that is already used). This is for consistency with other OpenStack APIs.

Alternatives

None

Data model impact

There should be a 1:1 mapping between a <logical name> and a <node uuid>. We can consider a <logical name> as an alias for a <node uuid>.

When the version of ironic that includes these changes is run against an existing installation, the database will be upgraded to support the addition of the <logical name> field to the node object.

The association between logical-name and UUID will need to be stored as a new attribute on the node object.

In the case where no logical name has been set, this field will be None.

What is a logical name?

This change introduces the concept of a <logical name> to ironic so that a human readable name can be associated with nodes. This <logical name> should be hostname safe, that is, the node logical name should also be usable as the hostname for the instance. For this to be true, the following references should be used to define what is a valid <logical name>: [wikipedia:hostname], [RFC952] and [RFC1123].

In simple english, what this means is that <logical names>s can be between 1 and 63 characters long, with the valid characters being [a-z0-9] and -, except that a <logical name> cannot begin or end with a -.

As a regular expression, this can be represented as: <logical name> == [a-z0-9]([a-z0-9-]{0,61}[a-z0-9]|[a-z0-9]{0,62})?

Note: It is recognised that a valid <logical name> could also be a valid <node uuid>, which could lead to confusion. As a consequence, logical names will be rejected as invalid if they are valid UUIDs.

Search Ordering (implementation hint)

As we now have two options to specify a node - the logical name and the node uuid - it is suggested that the following search order be implemented to define the behaviour of ironic. The following pseudo-code is provided to assist implementation:

if is_uuid_like(value):
    # handle it like a UUID

else if is_hostname_like(value):
    # handle it like a logical name

else:
    # invalid format, raise error

REST API impact

A number of existing APIs will need to be modified to support the use of <logical_name>.

The following APIs will add in a new JSON body parameter named name:

- DELETE /v1/nodes
- PATCH /v1/nodes
- GET /v1/nodes/validate

The following APIs will add in a new response body field named name:

- GET /v1/nodes

The following APIs will reflect the existing node_uuid version of this API, along with adding support to specify logical_name instead of node_uuid in the URL:

- GET /v1/nodes/(node_logical_name)
- PUT /v1/nodes/(node_logical_name)/maintenance
- DELETE /v1/nodes/(node_logical_name)/maintenance
- GET /v1/nodes/(node_logical_name)/management/boot_device
- PUT /v1/nodes/(node_logical_name)/management/boot_device
- GET /v1/nodes/(node_logical_name)/management/boot_device/supported
- GET /v1/nodes/(node_logical_name)/states
- PUT /v1/nodes/(node_logical_name)/states/power
- PUT /v1/nodes/(node_logical_name)/states/provision
- GET /v1/nodes/(node_logical_name)/states/console
- PUT /v1/nodes/(node_logical_name)/states/console
- POST /v1/nodes/(node_logical_name)/vendor_passthru

RPC API impact

None

Driver API impact

None

Nova driver impact

This change as specified here is wholly contained with ironic itself. It is most probably beneficial to expose the concept of a logical name to outside ironic for use in the Nova API.

If required, this will be addressed in an independent spec.

Security impact

None

Other end user impact

If Horizon allows a user to enter a node UUID, and validates it as conforming to a particular regex, then this will most likely require change to support either a <node uuid> or <logical name>.

python-ironicclient

In each sub-command in python-ironicclient where a node UIUD can be specified, we will need to be able to support a logical name in its place. Please see the detailed changes in the REST API section for an idea of the scope of change required.

Scalability impact

None

Performance Impact

None

Other deployer impact

None

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
mrda - Michael Davies <michael@the-davies.net>

Work Items

1. REST API additions and modifications
2. python-ironicclient additions and modifications

Dependencies

None

Testing

Unit testing will be sufficient to verify the veracity of this change

Upgrades and Backwards Compatibility

As mentioned above, when the code that includes this change is run against a previous Ironic install that does not have this change, the database will need to have its schema updated to add in the additional field name.

Documentation Impact

Online documentation for both the Ironic API and python-ironicclient will need to be updated to accompany this change.

References

The need for this change was discussed at the Kilo Summit in Paris (ref https://etherpad.openstack.org/p/kilo-ironic-making-it-simple)

- [wikipedia:hostname] - http://en.wikipedia.org/wiki/Hostname
- [RFC952] - http://tools.ietf.org/html/rfc952

- [RFC1123] - http://tools.ietf.org/html/rfc1123

5.14.19 Add maintenance reason field

https://blueprints.launchpad.net/ironic/+spec/maintenance-reason

When a node is put into maintenance (manually or automatically), Ironic and the operator should know why.

Problem description

Ironic has the ability to mark a node in maintenance mode, to be ignored for the purposes of scheduling and verifying state. However:

- When Ironic automatically puts a node into maintenance mode, it sets the reason in the *last_error* field, which may get overwritten by other tasks later.
- When an operator manually puts a node into maintenance mode, they have no method to show why it was put into maintenance, for other operators or to remind themselves later.

Proposed change

The following should be enough to solve this problem:

- A *maintenance_reason* field should be added to the nodes table, as the canonical place to store the reason the node was put into maintenance mode. This should be an internal attribute not directly editable by calling the node.update API.
- A new API endpoint should be added to more easily manage maintenance mode. This endpoint can toggle maintenance mode on or off, with an optional reason for on, and clearing the reason when toggled off. Changing maintenance mode using the old methods should still be allowed for backwards compatibility.
- Modify node.update to clear the maintenance reason when turning maintenance mode off via node.update API.

Alternatives

Alternatively, operators could store this in another system, such as a CMDB.

While I think this would be fine, this would not allow for Ironic to automatically set a maintenance reason when putting a node into maintenance mode. Work would need to be done to make Ironic notify the operator or integrate with the other system; and possibly cause the operator to do manual work to put the reason in the other system.

Data model impact

This will add a *maintenance_reason* field to the *node* table, with an accompanying database migration. This field will default to NULL, which will also be the value when there is no reason, or when maintenance reason is cleared via the new API.

REST API impact

One new endpoint will be added, with two methods:

- PUT /v1/nodes/<uuid>/maintenance

  - Puts a node into maintenance mode, with an optional reason.
  - Method type: PUT
  - Normal response code: 202
  - Expected errors:
    * 404 if the node with <uuid> does not exist.
    * 400 if a conductor for the nodes driver cannot be found.
  - URL: /v1/nodes/<uuid>/maintenance
  - URL parameters: None.
  - JSON body: {reason: Some reason.}, or {} or empty for no reason.
  - Response body is empty if successful.
- DELETE /v1/nodes/<uuid>/maintenance
  - Takes a node out of maintenance mode and clears the reason.
  - Method type: DELETE
  - Normal response code: 202
  - Expected errors:
    * 404 if the node with <uuid> does not exist.
    * 400 if a conductor for the nodes driver cannot be found.
  - URL: /v1/nodes/<uuid>/maintenance
  - URL parameters: None.
  - JSON body: None.
  - Response body is empty if successful.

The *maintenance_reason* field should be added to the node details API.

RPC API impact

None.

Driver API impact

None.

Nova driver impact

None.

Security impact

None.

Other end user impact

Support for this will be added in python-ironicclient. The CLI will look like:

```
usage: ironic node-set-maintenance [--reason <reason>]
                                   <node id> <maintenance mode>

Set maintenance mode on or off.

Positional arguments:
  <node id>           UUID of node
  <maintenance mode>  Supported states: 'on' or 'off'

Optional arguments:
  --reason <reason>   The reason for setting maintenance mode to "on"; not
                      valid when setting to "off".
```

Scalability impact

None.

Performance Impact

None.

Other deployer impact

Deployers may wish to start using this feature when it is deployed; however there should be no impact otherwise.

Developer impact

None.

Implementation

Assignee(s)

Primary assignee:
jroll

Other contributors:
lucasagomes

Work Items

- Add *maintenance_reason* to the nodes table with a migration.
- Set *maintenance_reason* when automatically setting maintenance mode.
- Add the new API endpoints.
- Clear maintenance_reason when using node.update to set maintenance mode off.
- Add client support for the new API endpoints.

- Add Tempest tests for the new API endpoints.

Dependencies

None.

Testing

Tempest tests should be added for the new API endpoints.

Upgrades and Backwards Compatibility

This change will be backwards compatible with existing clients, as they may still use the node.update call to set maintenance on or off. Updating via the node.update call will not be deprecated in v1, since there isnt any reasonable programmatic way to inform users of its deprecation. It will be deprecated in v2.

To avoid having an outdated maintenance reason, using the node.update call to set maintenance mode off will clear the maintenance reason.

Documentation Impact

The new API endpoints and client methods should be documented.

References

None.

5.14.20 A proposal for the New Ironic State Machine.

https://blueprints.launchpad.net/ironic/+spec/new-ironic-state-machine

This blueprint suggests reworking the Ironic provisioning state machine to fix some current shortcomings and to make it easier for drivers and external orchestration agents to manage nodes in Ironic.

NOTE: This blueprint describes the functionality we intend the new state machine to have. Actual implementation of this spec, including detailed upgrade paths and technical arcana will be handled by other specs.

Problem description

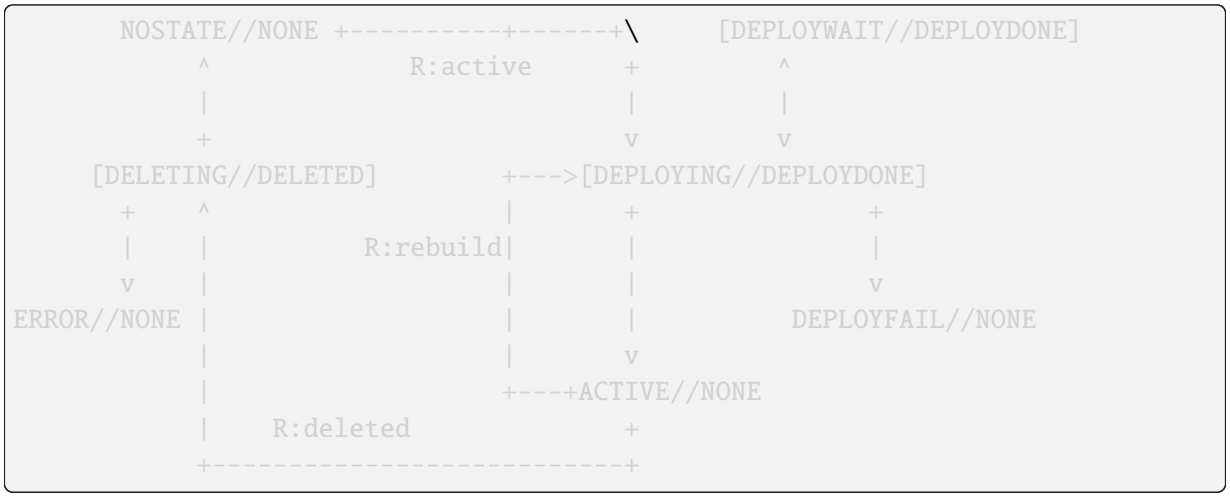
The current Ironic state machine has a few shortcomings:

- NOSTATE is a state that indicates we have no state information about a node. This may be fine for talking about the nodes power state, but we should always know what provisioning state a node is in.
- We also need a state to put nodes in when they are performing configuration tasks that can reasonably be expected to take hours to complete, such as RAID configuration and burn in. It is unreasonable to force upstream consumers of Ironic managed nodes to wait hours between the time they request a node and get it, so running these tasks as part of DEPLOYING or DEPLOYWAIT is nonviable.
- We also need a place to handle node decommissioning tasks. The current decommissioning blueprints handle this add decommissioning and decommissioned states, but it would also be useful to perform decommissioning tasks on freshly-added nodes.

- We also need to let external orchestration systems hook into parts of the state machine for each node to let them manage parts of the node life cycle without having to import that functionality into Ironic. Details of how that will happen will be covered in a different spec.

Proposed change

Current state machine:



Legend for the current state machine:

- [STATE] indicates an active state. Ironic is doing something to the node.
- STATE indicates a stable (or passive) state. Ironic will not transition unless receiving a request via the API.
- R:request indicates the request which must be passed to the API to initiate a transition out of a stable state.

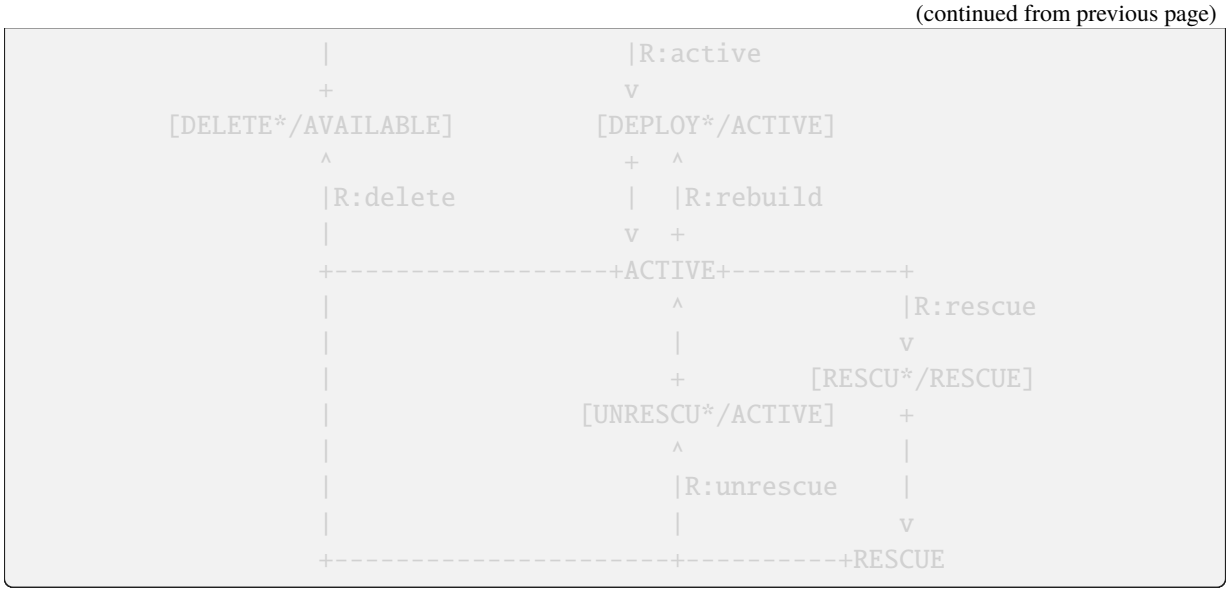
Ironics API presents two fields for the provision_state of a node: current and target. Thus, in this diagram, all states are represented as CURRENT-slash-slash-TARGET state.

Descriptions of the states for the current state machine can be found *here <https://github.com/openstack/ironic/blob/stable/icehouse/ironic/common/states.py>*.

New state machine:

(continues on next page)

(continued from previous page)



Legend for the new state machine:

[STATE*/TARGET]
: STATE* indicates an active state, a momentary state, and a fail state. The active state has an -ING suffix, the momentary state has a -ED suffix, and the fail state has a -FAIL suffix. In the active state, Ironic is doing something to the node.

  - If the steps taken during the active (-ING) state succeed, Ironic will automatically transition to the momentary (-ED) state and then to the next indicated state on the graph. Unless there are special rules for momentary states, they will not be separately described.
  - If it fails, Ironic will transition to the fail (-FAIL) state. Unless there are special rules for the fail state, it will not be separately described.

  TARGET indicates the target state that Ironic will try to transition the node to from the active state. TARGET must be a stable state.

STATE
: A stable (or passive) state, usually the target of a particular set of state transitions. Ironic will not transition away from this state without an API request to do so.

R:request
: Indicates that the transition so labeled happens as a result of this particular API call.

Descriptions of the new states:

ENROLL
: This is the state that all nodes start off in. When a node is in ENROLL, the only thing Ironic knows about it is that it exists, and Ironic cannot take any further action by itself. Once a node has its drivers and the required information for each driver in node.properties, the node can be transitioned to VERIFYING via the manage API call

VERIFYING
: Ironic will validate that it can manage the node with the drivers and the credentials it has been assigned. For drivers that manage power state of the node, this must involve actually going out and confirming that the credentials work to access whatever node control mechanism they talk to.

MANAGEABLE
: Once Ironic has verified that it can manage the node using the driver and credentials passed in at

node create time, the node will be transitioned to MANAGEABLE and (optionally) powered off. From MANAGEABLE, nodes can transition to:

- MANAGEABLE (through CLEANING) via the clean API call,
- MANAGEABLE (through INSPECTING) via the inspect API call, and
- AVAILABLE (through CLEANING) via the provide API call.

INSPECTING
INSPECTING will utilize node introspection to update hardware-derived node properties to reflect the current state of the hardware. We expect this state to get its data via the driver introspection interface (reference to spec forthcoming). If introspection fails, the node will transition to INSPECTFAIL.

CLEANING
Nodes in the CLEANING state are being scrubbed in preparation to being made AVAILABLE. Good candidates for CLEANING tasks include:

- Erasing the drives.
- Validating firmware integrity.
- Verifying that the actual hardware configuration matches what is described in node.properties.
- Booting to a long running deploy ramdisk, if you want the machine to stay on while in AVAILABLE.

No matter what tasks are performed during CLEANING, the apparent configuration of the system must not change. For instance, if you tear down a set of RAID volumes to securely erase each physical disk separately, you must rebuild the RAID volumes you tore down.

When a node is in CLEANING state it means that the conductor is executing the clean step (out-of-band) or preparing the environment (building PXE configuration files, configuring the DHCP, etc..) to boot the ramdisk.

CLEANWAIT
Just like the CLEANING state, the nodes in CLEANWAIT are being prepared to become AVAILABLE. The difference is that in CLEANWAIT the conductor is waiting for the ramdisk to boot or the clean step which is running in-band to finish.

The cleaning process of a node in CLEANWAIT can be interrupted via the `abort` API call.

AVAILABLE
Nodes in the AVAILABLE state are cleaned, preconfigured, and ready to be provisioned. From AVAILABLE, nodes can transition to:

- ACTIVE (through DEPLOYING) via the active API call.
- MANAGEABLE via the manage API call

DEPLOYING
Nodes in DEPLOYING are being actively prepared to run a workload on them. This should mainly consist of running a series of short-lived tasks, such as:

- Setting appropriate BIOS configurations
- Partitioning drives and laying down file systems.

- Creating any additional resources (node-specific network config, etc.) that may be required by additional subsystems.

Tasks for DEPLOYING should be handled in a manner similar to how they are handled for CLEANING (details to be addressed in a different spec).

DEPLOYWAIT
Just like the DEPLOYING state, the nodes in DEPLOYWAIT are being deployed. The difference is that in DEPLOYWAIT the conductor is waiting for the ramdisk to boot or execute parts of the deployment which needs to run in-band on the node (for example, installing the bootloader, writing the image to the disk when iSCSI is not used, etc).

The deployment of a node in DEPLOYWAIT provision state can be interrupted via the `deleted` API call.

ACTIVE
Nodes in ACTIVE have a workload running on them. Ironic may collect out-of-band sensor information (including power state) on a regular basis, but will otherwise leave them alone. Nodes in ACTIVE can transition to:

- RESCUE (through RESCUING) via the rescue API call,
- AVAILABLE (through DELETING and CLEANING) via the delete API call, or
- ACTIVE (through DEPLOYING) via the rebuild API call.

RESCUING
Nodes in RESCUING are being booted into a temporary operating environment for troubleshooting or maintenance related reasons.

RESCUE
RESCUE exists to allow Ironic to be aware of a node that would be otherwise running a workload, but that is booted into a different operating environment for maintenance or troubleshooting reasons. From RESCUE, nodes can transition to:

- ACTIVE (through UNRESCUING) via the unrescue API call, or
- AVAILABLE (through DELETING and CLEANING) via the delete API call.

UNRESCUING
Nodes in UNRESCUING are being transitioned back to ACTIVE from RESCUE. Ironic will unwind whatever it needed to do to get the node into RESCUE

DELETING
Nodes in DELETING state are being torn down from running an active workload. In DELETING, Ironic should tear down or remove any configuration or resources it added in DEPLOYING.

Alternatives

No reasonable ones that we could think of at the summit.

Data model impact

Under the current state machine, NOSTATE is represented by a NULL in the database. This will require a database migration to change all NULLs to AVAILABLE along with special-case API handling during the migration. The additional states should not require changes to the data model.

REST API impact

We will provide the following verbs to manage the node lifecycle in the state machine:

Verb	Initial State	Intermediate States	End State
manage	ENROLL	VERIFYING -> VERIFIED	MANAGEABLE
clean	MANAGEABLE	CLEANING -> CLEANED	MANAGEABLE
inspect	MANAGEABLE	INSPECTING -> INSPECTED	MANAGEABLE
provide	MANAGEABLE	CLEANING -> CLEANED	AVAILABLE
manage	AVAILABLE	(none)	MANAGEABLE
active	AVAILABLE	DEPLOYING -> DEPLOYED	ACTIVE
rebuild	ACTIVE	DEPLOYING -> DEPLOYED	ACTIVE
rescue	ACTIVE	RESCUING -> RESCUED	RESCUE
unrescue	RESCUE	UNRESCUING -> UNRESCUED	ACTIVE
deleted	ACTIVE	DELETING -> DELETED -> CLEANING -> CLEANED	AVAILABLE
deleted	RESCUE	DELETING -> DELETED -> CLEANING -> CLEANED	AVAILABLE
deleted	DEPLOYWAIT	DELETING -> DELETED -> CLEANING -> CLEANED	AVAILABLE
abort	CLEANWAIT	(none)	CLEANFAIL

The API will remain backwards compatible with the active, rebuild, and delete verbs.

Unless otherwise required for backwards compatibility, the verbs must be called when the node is in the Initial State, and Ironic will perform all actions and transitions needed to move through the Intermediate States to the End State.

Since we are adding new states, older API clients may behave unexpectedly when they encounter a node in a state they do not understand.

RPC API impact

Not as a direct impact of this spec (beyond what is mentioned in the REST API impact section), but all the to-be-written specs which will actually implement the new states will have significant RPC and REST api impact.

Driver API impact

Yes. Large swaths of driver code will need a refactor to cooperate with the new per-node state machines.

Nova driver impact

NOSTATE has been renamed to AVAILABLE. This will require some glue code and creating an upgrade path.

Security impact

Probably not, assuming perfect coding.

Other end user impact

Yes.

Scalability impact

Probably nothing significant.

Performance Impact

Ditto.

Other deployer impact

Nodes will not automatically transition from ENROLL to MANAGEABLE. Deployers must assign drivers and add credentials to the node and then call the manage API before Ironic can manage the node.

Nodes will not automatically transition from MANAGEABLE to AVAILABLE, deployers will need to do that via the API before nodes can be scheduled.

Developer impact

Current and new Ironic drivers will need rework to comply with the new state machine.

Implementation

Assignee(s)

None yet.

Work Items

Specs need written to hash out the implementation details that the new state machine implies.

Dependencies

Most every blueprint that touches on the Ironic drivers will be affected, but this blueprint is vendor-agnostic.

Testing

None for this spec, but the implementation specs will need to address testing impacts of the changes they recommend.

Upgrades and Backwards Compatibility

None for this spec, but the implementation specs will need to address upgrade and backwards compatibility.

Documentation Impact

This spec should be used as initial documentation for the new state machine.

References

Anyone have a link to some developer session notes? I was sorta busy being a whiteboard monkey: https://i.imgur.com/tCxUCYk.jpg

5.14.21 Support for non-glance image references

https://blueprints.launchpad.net/ironic/+spec/non-glance-image-refs

Add the ability to provide non-Glance references for images that are used by Ironic.

Problem description

Currently, kernels and ramdisks, image_source images are downloaded from Glance, provided their UUID. This requires Glance to be up and running, and does not allow providing images from specific URL or from local disk.

Proposed change

The proposal is to create new image service base class for downloading images provided URL for them; also to add some common protocol support, i.e. downloading from remote HTTP servers and using images available in local file system.

Depending on the URL, different image service will be used:

- If URL starts with glance:// or is just an image UUID (for backwards compatibility), Glance image service is used.
- If it starts with http:// or https:// then it will be downloaded by HTTP image service.
- If it starts with file:// then it is referencing some file system available locally, either hard link will be created for an image if its in the same file system as folder with nodes images, or image will be copied to that folder by image service for current conductors local files.

Alternatives

Continue having a hard-dependency on Glance.

Data model impact

None

REST API impact

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Security impact

Operators should download images from trusted sources.

Other end user impact

None

Scalability impact

Depending on the protocol used to download image, network usage can be either reduced (using local files) or remains the same.

Performance Impact

None

Other deployer impact

Since Ironic may be used without Glance, developers cant make the assumption that Glance image metadata is the only source for such information. Deployers must be capable of supplying Ironic with all required metadata programmatically, and such requirements must be documented.

For example, because of kernel and ramdisk UUIDs are currently got from image_source image properties returned by Glance, links for those should be put into instance_info dictionary if Glance is not used. Another example is whole disk instance images that need to have is_whole_disk flag in instance_info in order to not to fetch kernel and ramdisk.

Developer impact

Developers can easily add their own image services to download images using specific protocols that are needed.

Implementation

Assignee(s)

Primary assignee:
vdrok

Other contributors:
None

Work Items

- Implement base image service class for downloading images from URL.
- Implement image service classes for downloading from HTTP server and from local file system.
- Implement a class that will return image service based on protocol defined in URL.

Dependencies

None

Testing

Tests for downloading images using different protocols will be added to Tempest.

Upgrades and Backwards Compatibility

For backwards compatibility it is allowed for image URL to contain only Glance image UUID.

Documentation Impact

Possibility of specifying URLs for images and protocols supported should be added to documentation.

If a driver uses some image metadata provided by Glance, it should be added to documentation, so that operators that decide not to run Glance can know which additional metadata they should provide manually.

References

https://etherpad.openstack.org/p/kilo-ironic-making-it-simple

5.14.22 Root device hints

https://blueprints.launchpad.net/ironic/+spec/root-device-hints

Allow operators to pass some hints to Ironic to decide which device should be selected for the deployment.

Problem description

When the deploy ramdisk boots Ironic picks the first disk it finds to be the root device (the device where the image will be put on). If the server has more than one SATA, SCSI or IDE disk controller, the order in which their corresponding device nodes are added is arbitrary [1] [2]. This may result in devices like /dev/sda and /dev/sdb switching around on each boot and Ironic picking different disk every time the machine is being deployed.

As an operator, if my server has multiple disks I would like to choose which one Ironic should deploy the image onto. Or in case I have created a RAID device to be my root device, Id like to tell Ironic to always use that.

Another problem, in case for the full disk image deployment, if we deploy a server twice and on each deployment Ironic picks a different disk we could end up with 2 disks containing a valid bootloader.

Proposed change

The change proposed by this blueprint is to give operators a means via the Ironic API to pass some hints about what disk should be picked in deploy time. That way Ironic can always pick the right disk to write the image on.

Also, with the addition of Ironic being able to create RAID arrays, it would be nice to be able to tell Ironic to use the device that was just created to be the root device for the deployment.

This spec is proposing having a limited number of hints that could be passed as part of the initial work, but could be extended later on. The initial proposed hint list is:

- model (STRING): device identifier
- vendor (STRING): device vendor
- serial (STRING): disk serial number
- wwn (STRING): unique storage identifier
- hctl (STRING): Host:Channel:Target:Lun for SCSI
- size (INT): size of the device in GB

The hints should live in the *properties* attribute of the Node resource, the key would be *root_device* and the value a dictionary so operators could combine one or more hints. For example:

```
node.properties['root_device'] = {'wwn': '0x4000cca77fc4dba1'}
```

The logic about which disk will be picked will follow:

1. If the hints are not specified Ironic will continue to pick the first disk it finds.
2. If hints are specified and only one disk is found Ironic will pick it.
3. If hints are specified and multiple disks are found Ironic will pick the first disk that matches the all the criteria.
4. If hints are specified and no disks are found the deployment is aborted.

The default deploy ramdisk and IPA needs to be changed to support filtering the disks based on the hints, if specified.

Alternatives

We could recommend operators to avoid having multiple storage devices on the machines being managed by Ironic.

Data model impact

None

REST API impact

As we want to use a dictionary as a value on the *properties* attribute the bug 1398350 needs to be fixed.

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

Deployers will have a finer granularity in selecting the disk device to be used for the deployment.

> **Note**
>
> When specifying device size as a hint operator needs to make sure that the value doesnt conflict with the local_gb properties of the node. This is going to be documented as part of this spec.

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
lucasagomes

Other contributors:
None

Work Items

- Make Ironic check for hints in the node.properties
- Pass the hint information to the deploy ramdisk and IPA
- Add tests and documentation
- Modify the default deploy ramdisk in diskimage-builder to consider the hints when picking the disk device
- Modify IPA to consider the hints when picking the disk device

Dependencies

- bug 1398350 needs to be fixed.

Testing

- Unit tests will be added

Upgrades and Backwards Compatibility

The change is backwards compatible since if hints are not specified Ironic will continue to do what it does today (pick the first disk it found for the deployment).

Documentation Impact

A document explaining how hints works and what are the options and values supported is going to be added.

References

None

5.14.23 Seamicro Serial Console

https://blueprints.launchpad.net/ironic/+spec/seamicro-serial-console

This blueprint implements console driver ShellinaboxConsole which support serial console access for Seamicro Fabric Compute system.

Problem description

Currently there is no support to get the serial console for physical server configured within the Seamicro Fabric Compute system.

Proposed change

Implements a console driver ShellinaboxConsole that uses telnet+shellinabox to connect to serial console of physical servers configured within the Seamicro Fabric Compute system. SeaMicro System provides telnet facility to connect to any of its physical servers serial console. Port to which we telnet depends on server-id. server-id here, is what SeaMicro box refers to its server and not ironic nodes uuid. We already have the required information (server-id) captured as part of driver_info. Following are details,

- Use existing ironic/drivers/modules/console_utils module to start/stop shellinabox. IPMI driver already use shellinabox to access their serial console.

- Add seamicro_terminal_port property to CONSOLE_PROPERTIES. This is going to be port on which shellinabox listens locally.
- Add new class ShellinaboxConsole inherited from base.ConsoleInterface in ironic/drivers/modules/seamicro.py
- **Implement the following methods in ShellinaboxConsole class.**
  - **`validate()` - Validate the Node console info.**
    - param task : a task from TaskManager.
    - raises : InvalidParameterValue
  - `start_console()` - Start a remote console for the node.
  - `stop_console()` - Stop the remote console session for the node.
  - `get_console()` - Get the type and connection information about the console.
- Add self variable self.console in PXEAndSeaMicroDriver.

Alternatives

None

Data model impact

None

REST API impact

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

The following which are already part of driver_info fields are required:

- `seamicro_api_endpoint` - hostname or IP address of seamicro
- `seamicro_username` - seamicro username
- `seamicro_password` - seamicro password
- `seamicro_server_id` - seamicro server id

Additionally one field need to be provided with driver_info

- `seamicro_terminal_port` - nodes UDP port to connect to. Only required for console access.

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
harshada-kakad

Work Items

Implement `ShellinaboxConsole` class inherited from `base.ManagementInterface`. Implement `validate` , `start_console`, `stop_console`, `get_console`.

Dependencies

None

Testing

Unit Testing will be added.

Upgrades and Backwards Compatibility

None

Documentation Impact

As part of this blueprint I would be documenting usage of this feature.

References

None

5.14.24 UEFI Secure Boot support for iLO drivers

https://blueprints.launchpad.net/ironic/+spec/uefi-secure-boot

Some of the Ironic deploy drivers support UEFI boot. It would be useful to security sensitive users to deploy more securely using Secure Boot feature of the UEFI. This spec proposes alternatives to support Secure Boot in baremetal provisioning for iLO drivers.

Problem description

Secure Boot is part of the UEFI specification (http://www.uefi.org). It helps to make sure that node boots using only software that is trusted by Admin/End user.

Secure Boot is different from TPM (Trusted Platform Module). TPM is a standard for a secure cryptoprocessor, which is dedicated microprocessor designed to secure hardware by integrating cryptographic keys into devices. Secure Boot is part of UEFI specification, which can secure the boot process by preventing the loading of drivers or OS loaders that are not signed with an acceptable digital signature.

When the node starts with secure boot enabled, system firmware checks the signature of each piece of boot software, including firmware drivers (Option ROMs), boot loaders and the operating system. If the signatures are good, the node boots, and the firmware gives control to the operating system.

The Admin and End users having security sensitivity with respect to baremetal provisioning owing to the workloads they intend to run on the provisioned nodes would be interested in using secure boot provided by UEFI.

Once secure boot is enabled for a node, it cannot boot using unsigned boot images. Hence it is important to use signed bootloaders and kernel if node were to be booted using secure boot.

Proposed change

Preparing the environment

- The operator informs the Ironic using the `capabilities` property of the node. The operator may add a new capability `secure_boot=True` in `capabilities` within `properties` of that node. This is an optional property that can be used if node needs to be provisioned for secure boot. By default the behavior would be as if this property is set to False. The iLO hardware discovery feature (proposed) could auto discover the secure boot capability of the node and create node capability into that node object in future.
- If the user has `secure_boot` capability set in the flavor, iLO drivers have ability to change the boot mode to UEFI and prepare the node for the secure boot on the fly using proliantutil library calls.

Preparing flavor for secure boot

- The `extra_specs` field in the nova flavor should be used to indicate secure boot. User will need to create a flavor by adding capabilities:secure_boot=True to it.
- iLO driver will not do secure boot if secure_boot capability flavor is not present or set to False. Nova scheduler will use secure_boot capability as one of the node selection criteria if secure_boot is present in extra_spec. If secure_boot is not present in extra_spec then Nova scheduler will not consider secure_boot capability as a node selection criteria.
- Ironic virt Driver needs to pass the flavor capability information to the driver as part of instance info. Having capability information as part of instance info would help driver in preparing and decommissioning the node appropriately. With respect to secure boot feature, instance info should contain the capability info related to `secure_boot`. This information would be used by iLO driver for :-
  - **During provisioning, driver can turn on the secure boot capability to** validate signatures of bootloaders and kernel.
  - During cleaning stage of teardown, clean_step could be added to initiate steps to clear the signatures, if any stored onto the node signature database.

Preparing boot and deploy images

Disk Image builder changes are required to integrate signed shim and grub bootloaders. shim bootloader is required as it is signed using Microsoft UEFI CA signature and recognises corresponding linux vendors certificate as a valid certificate. Secure boot enabled Proliant UEFI systems are pre-loaded with Microsoft UEFI CA signatures. User signed images can be supported but users needs to manually configure their keys to system ROM database using Proliant tools.

Alternatives

None

Data model impact

None

REST API impact

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Security impact

This enhances security. Only correctly signed firmware, bootloader and OS can be booted. It provides users with the opportunity to run the software of their choice in the most secure manner.

Other end user impact

Users need to use properly signed deploy and boot components. Currently iLO driver would support deploy and boot images having shim and grub signed by Linux OS vendors. If user wants to use custom signed images, then he would need to manually configure their keys to UEFI using Proliant tools.

Scalability impact

None

Performance Impact

There is no performance impact due to signature validation in secure boot.

Other deployer impact

User can deploy only signed images with secure boot enabled. If the user wants to use custom unsigned images for decommissioning then he would need to disable secure boot on the node as part of clean_step during teardown stage before booting into such custom images.

Developer impact

None

Implementation

Assignee(s)

primary author and contact.

Primary assignee:
Shivanand Tendulker (stendulker@gmail.com)

Work Items

1. Implement code changes for supporting secure boot.
3. Implement secure boot iLO drivers.
4. Changes into Nova Virt Driver to pass capability information in the flavor as instance info. It is being proposed as part of following design spec. https://review.openstack.org/136104

Dependencies

1. DIB changes are required to add signed shim and grub2 to the ubuntu cloud image creation using disk-image-create and ramdisk-image-create scripts.
2. Changes in Nova Virt driver to pass capabality information from flavor to driver through instance info.

Testing

Unit tests would be added for all newly added code.

Upgrades and Backwards Compatibility

None

Documentation Impact

Newly added functionality would be appropriately documented.

References

Discover node properties for iLO drivers https://review.openstack.org/#/c/103007

Ironic Management Interfaces to support UEFI Secure Boot https://review.openstack.org/#/c/135845

5.14.25 Whole Disk Image Support

https://blueprints.launchpad.net/ironic/+spec/whole-disk-image-support

This spec proposes to add a feature of deploying whole disk images to Ironic.

Problem description

Currently, the Ironic PXE deploy driver and the iLO deploy driver deploys only partition images making an Image kernel/ramdisk to be mandatory. The current approach makes it impossible to deploy Images that are not capable of providing a corresponding kernel/ramdisk. A significantly important use-case would be to deploy Windows Images on baremetal systems.

Proposed change

Ironics deploy drivers will infer if they have to deploy a whole disk image or a partition image based on the presence of a kernel/ramdisk by querying Glances properties.

To utilise the scheduler efficiently, Ironic deployments of whole disk images will only accept a root-only flavor to efficiently utilise the entire disk. Any other flavor type would be rejected during the validation phase in Ironic. To help the scheduler fail fast, a new filter will be added to the scheduler which will compare the image structure with the flavor attributes to check if it can proceed with scheduling.

For the PXE Deploy driver, once the image structure is inferred and is found out to be a whole disk image, the image is dumped onto the disk-lun and the node is restarted with a pxe config file that instructs the server to boot from the local disk(PXE-localboot).

The agent driver currently only supports deploying whole disk images, however, the agent driver will adopt the inference pattern stated above to deploy whole disk images.

The iLO virtual media iscsi deploy driver needs to be validated to deploy whole disk images which will use the same mechanism the pxe driver uses to write whole disk images.

Alternatives

Having an optional is_whole_disk_image property explicitly for a Glance image and using that value to figure out if the deploy driver should deploy a whole disk image or not.

It was suggested that one could assume image type(i.e part or disk) based on the image format. So, AMI for Partition images, QCOW2, RAW, etc for Disk Images. This does not seem appropriate as the other image formats could also very well be used as a Partition image with a certain Kernel/Ramdisk.

Data model impact

None

REST API impact

None

RPC API impact

None

Driver API impact

None

Nova driver impact

A separate Ironic-only Nova filter will be added which will validate flavor attributes against image structure.

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

For those using disk-image-builder to build images, currently, the *vm* element should help in building of whole-disk-images.

Developer impact

The other deploy drivers need to keep in mind of the pattern being used currently to infer deployment of whole disk images while writing their own logic for the same feature.

Implementation

Assignee(s)

Primary assignee:
sirushtim

Work Items

Modify PXE Deploy Driver to support deploying of whole disk images.

Modify Agent Deploy Driver to use the whole-disk-image inference pattern since it already supports deploying of whole-disk-images by default.

Modify iLO virtual media Deploy Driver to support deploying of whole disk images.

Add a Nova filter to validate image structure against flavor attributes.

Dependencies

None

Testing

Tempest tests need to be added to validate deployment of Disk Images.

The Cirros whole disk image will be used to test the deployment of whole disk images in Ironic.

Upgrades and Backwards Compatibility

None

Documentation Impact

Add user-facing docs to explain how whole disk images should be deployed via Ironic.

References

https://etherpad.openstack.org/p/icehouse-ironic-windows-support

5.15 Juno

5.15.1 Add instance_info field to Node model

https://blueprints.launchpad.net/ironic/+spec/add-node-instance-info

This blueprint will introduce a new field to the Nodes resource that will be used to store instance-specific data, which will be used by the driver when provisioning or managing that instance.

Problem description

The metadata which describes a particular instance being deployed is being passed from the client to the deploy driver via the node.driver_info field. That field is intended to store only the driver-specific metadata needed by that driver for the purpose of managing and provisioning that specific node (such as IPMI credentials). Such metadata should be constant across multiple deployments to the node, and updating the driver_info often corresponds to an operator action (such as rotating passwords).

The instance-specific metadata, which changes during every deploy, should not be stored alongside the driver-specific metadata.

Proposed change

- create a new instance_info attribute to the Nodes resource where all the instance-level related data should live.
- modify the Nova Ironic driver to populate the instance_info instead of the driver_info when deploying a node.
- deploy ramdisk and deploy kernel to not be part of the Flavors extra_data and live in the Nodes driver_info instead.
- Remove all the data stored in the instance_info field as part of the nodes tear down process (Since all the data stored there will be related to the instance).

Alternatives

Continue to use the driver_info field?

Data model impact

The instance_info field will be added to the nodes table.

REST API impact

A new instance_info attribute will be added to the Node resource.

Driver API impact

None

Nova driver impact

When preparing the node to be deployed the Nova driver was adding the instance information to the nodes driver_info field, with this change the fields root_gb, swap_gb, ephemeral_gb, ephemeral_format, image_source will now be added to the instance_info field instead.

The nova driver also needs to be changed to not look at the flavors extra spec to get the deploy ramdisk and deploy kernel, such fields will be part of the process of enrolling a node and should be present at the node before triggering it to be deployed (otherwise it will fail in the validation).

Security impact

As a side effect, this change enables a later improvement in security by allowing the driver_info field to be hidden from the REST API, if policies were added and enforced in the API layer. This would allow the management credentials to be hidden from users.

Other end user impact

Instead of setting the deploy ramdisk and deploy kernel as part of the flavor in Nova, this fields if required (depending on the driver) should be set by the operator in the driver_info field of the Nodes when enrolling it.

A migration script will be created to demonstrate how to extract the deploy ramdisk and deploy kernel from the Nova flavor and update the Ironic Nodes driver_info field appropriately.

Scalability impact

None

Performance Impact

None

Other deployer impact

None

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
lucasagomes

Other contributors:
jroll

Work Items

- create instance_info field in the nodes table in the database.
- create a new instance_info attribute in the Nodes object.
- modify the PXE driver to look at the new instance_info field to get the instance-level data when deploying a node.
- modify the Nova Ironic Driver to populate the Nodes instance_info field instead of the driver_info.
- modify the Nova Ironic Driver to not get the deploy ramdisk and deploy kernel from the flavor and populate in the Node before the deployment, those fields should be present in the Node already once its enrolled, validation should fail otherwise.

Dependencies

None

Testing

Unit tests and DevStack needs to be changed to cover the new changes.

The DevStack change will need to be staggered in three changes:

- have DevStack add the deploy kernel and deploy ramdisk to driver_info (while still also adding on flavor).
- land patch in Ironic.
- have DevStack stop writing deploy kernel and deploy ramdisk to flavor.

Documentation Impact

Documentation should be modified to instruct operators not to add the deploy kernel and ramdisk glance image UUIDs to the Nova flavor. Instead, the documentation will indicate that operators must pass this information to Ironic when enrolling nodes.

References

None

5.15.2 Add Support for Retry on NodeLocked Exceptions

https://blueprints.launchpad.net/ironic/+spec/add-nodelocked-retry

Lets reduce the pain of clients being presented errors due to conflicts with locking a particular node (NodeLocked exceptions) by adding multiple attempts to lock the node on the clients behalf.

As an added benefit, this would also help with tempest testing where this error is seen on occasion [1].

Problem description

Ironic clients may sometimes experience a NodeLocked exception/error when making certain REST API calls. This is because the conductor(s) will grab a lock on a node before performing any action that may change that nodes characteristics. Examples of where node locking occurs are:

- Almost all of the RPC API calls (called to satisfy some REST API requests) lock the node in question except for: change_node_maintenance_mode, driver_vendor_passthru, get_console_information and validate_driver_interfaces.
- The conductor periodic task to synchronize power states.
- The conductor periodic task to check deploy timeouts.

The amount of time the node locks are held are typically not long. We could eliminate many of these errors by simply retrying the lock attempt. Admittedly, this would not *totally* eliminate the problem, but it would make it much less likely to occur for clients, and thus make for a much better experience.

Proposed change

The TaskManager class is used to control when nodes are locked. The lock itself is implemented by the database backend.

I propose we change the TaskManager.__init__() method to incorporate the retry logic, leaving the implementation of the lock itself (the database API layer) untouched. This lets us change the lock implementation later, if we choose, without having to migrate the retry logic.

Alternatives

A more permanent solution to this problem is being discussed in a spec that is defining an asynchronous REST API [2]:

https://review.openstack.org/94923

It is unlikely that spec will be approved in Juno because of the work that it entails to change the APIs. This proposal is simple and can be implemented quickly.

A second alternative would be to implement the retry logic at the RPC level. This has the disadvantage of increasing traffic on the RPC bus, and even lengthier waits for lock attempts.

A third alternative is to change the locking layer itself and provide it a value for timing out the attempt. This could potentially be a more complex change, and would need to be duplicated if we changed the mechanism used for locking.

Data model impact

None

REST API impact

None

Driver API impact

None

Nova driver impact

None

Security impact

This could create a DoS opportunity based on configuration values for retry attempts and time between retries. This is eliminated by using configuration values that will force only a single lock attempt.

Other end user impact

The only impact this will have on the user is the reduced amount of node lock errors from API requests and potential delays in HTTP responses. See the Performance Impact section for more information on delays.

Scalability impact

This does have the possibility to negatively impact scalability. A spawned worker thread within the conductor could potentially take longer to process the work if a NodeLocked exception is thrown. This impact can be mitigated by increasing the number of workers in the pool (the workers_pool_size option).

Performance Impact

This will add additional processing time to REST API calls that happen to encounter a node lock error. This is due to repeated calls to the database API layer to attempt to successfully lock the node.

If the node is locked successfully on the first attempt, then performance is not impacted at all.

Other deployer impact

We should control the retry logic with configuration variables for maximum retry attempts and time in between attempts. Using sane defaults (setting the values such that no retry attempts are performed) should help alleviate much of this impact.

The following new configuration variables are proposed (and their default values) to be added to the conductor variable group:

- node_locked_retry_attempts = 3 (default to 3 retry attempts)
- node_locked_retry_interval = 1 (default to 1 second between attempts)

The default for node_locked_retry_attempts will be 3, which could potentially affect existing installations (see Performance Impact section) when upgraded, but should reduce NodeLocked errors encountered.

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
<dshrews>

Work Items

None

Dependencies

The retrying [3] Python library would be of great use here, as it encasulates all of the logic we would want. Version 1.2.2 (the latest release as of this writing) would be the minimum version we would want since it contains an important bug fix related to retrying on certain exceptions.

NOTE: The global-requirements.txt value [4] for the retrying module will need to be modified to meet this minimum version requirement.

Testing

I dont see how to test this in tempest successfully (other than eliminating the current errors from tempest due to this problem), but I imagine we can add unit tests to verify its working as we expect.

Documentation Impact

None

References

[1] https://bugs.launchpad.net/ironic/+bug/1321494 [2] https://review.openstack.org/94923 [3] https://pypi.python.org/pypi/retrying [4] https://github.com/openstack/requirements

5.15.3 Agent Driver

https://blueprints.launchpad.net/ironic/+spec/agent-driver

Ironic needs a deploy driver to interact with Ironic Python Agent.

Problem description

Today, Ironic is limited in the tasks that may be performed on a bare metal node. Actions like update firmware and secure erase disks are not possible. This is not due to any flaw in Ironic itself, but rather a limitation of the deploy ramdisk used by Ironics PXE driver.

Ironic Python Agent (IPA) is a project to provide a deploy agent for use by Ironic. This agent is designed to run in a ramdisk and perform maintenance on bare metal nodes, including hardware configuration and management, provisioning, and decommissioning of servers. This agent exposes a REST API that Ironic could call in order to perform various tasks.

Potential use cases:

- It is usually advantageous for a server to be running the latest BIOS firmware. Updating firmware is a critical feature in Ironic.
- End users often prefer that their data is erased when a server is released. The ability to run secure erase on a bare metal nodes disks is another critical feature for Ironic.
- End users may have varying workloads they wish to deploy to a bare metal node. Some users may wish to run an application on bare metal, where others may wish to run a hypervisor. These different workloads may require different BIOS configurations, such as switching the VT bit on or off. This utility ramdisk enables Ironic to manage these configurations.
- Some users may desire faster boot times for a single bare metal node. The utility ramdisk may be used to accomplish this by leaving the node running the ramdisk, ready for deployment. This removes one POST cycle from the deployment process, compared to the PXE driver. Additionally, the ramdisk could write popular images to the boot device before deployment time, to remove the time spent writing the image.
- End users may wish to use cloud-init with a configdrive to load data such as SSH keys or network configurations. The ramdisk can write a partition containing the configdrive to be used by the end user and their image.

Ironic needs a deploy driver that interacts with the agent, rather than the existing deploy ramdisk.

Proposed change

A full deploy driver that interacts with the agent should be implemented.

The driver will:

- Allow the agent to look up the UUID stored by Ironic for the node that the agent is running on.
- Allow the agent to periodically heartbeat. The driver should use this heartbeat to verify that the agent is online.
- Leverage the periodic heartbeat as a callback mechanism.
- Make calls to the agents REST API to instruct the agent to do deploy-related tasks, such as writing an image.
- Make calls to the agents REST API to perform decommissioning tasks.
- Behave similarly to the existing PXE driver, with an explicit goal of eventually merging the two (not a hard requirement, if this turns out to be impossible, this goal may be dropped).

Alternatives

The only alternative to building this driver would be to continue work on the existing PXE deploy driver, to deliver the functionality identified in this spec. Today, this driver has a very different model to the proposed agent driver. It seems best, from discussion with Ironics leadership, to start the work diverged and work from both ends to converge on a single driver.

Data model impact

- Two fields will exist on the *driver_info* field: agent_url and agent_last_heartbeat.

REST API impact

Two vendor_passthru methods will be added:

Node lookup method:

- Description: The agent will post a JSON blob containing detailed hardware information to this endpoint at startup. Ironic will use this information to determine which node the agent is running on (first iteration will look for matching MAC addresses), and pass the nodes UUID back, along with an integer (in seconds) defining the timeout for receiving another heartbeat from the agent.
- Method type: POST
- Normal response code: 200
- Expected errors:
  - 400: Invalid hardware data structure version sent.
  - 404: A node with the provided hardware information could not be found.
- URL: /{api_version}/drivers/{driver}/vendor_passthru/lookup
- Parameters: none.
- Body JSON schema:

```
{
  "version": 2
  "inventory": {
    "interfaces": [{...}, ...],
    "cpu": {...},
    "disks": [{...}, ...],
    "memory": {...}
  }
}
```

- Response JSON schema:

```
{
  "heartbeat_timeout": 300,
  "node": {
    "uuid": "some-uuid"
  }
}
```

Heartbeat method:

- Description: The agent will periodically send a heartbeat to Ironic to signal that it is still running, as well as immediately after completing a command from Ironic. The agent driver will leverage this heartbeat as a callback mechanism. If a node is powered on, not provisioned, and a heartbeat is not received within a configurable timeout period, then Ironic will take action on this node; perhaps attempt a reboot or put the node into maintenance mode. As part of the heartbeat request, the agent provides its endpoint URL, where Ironic can issue requests to the agent. Ironic stores the time of the heartbeat and the agents URL in *Node.driver_info*.
- Method type: POST
- Normal response code: 202
- Expected errors: * 404: The specified node could not be found.
- URL: /{api_version}/nodes/{node_uuid}/vendor_passthru/heartbeat
- Parameters: The nodes UUID is part of the URL.
- Body JSON schema:

```
{
  "agent_url": "http://1.2.3.4:9999/"
}
```

- Response JSON schema: none

Driver API impact

There is no impact on the driver API.

Nova driver impact

There is no impact on the Nova driver.

Security impact

Some authentication method will need to be implemented. Today, Ironics PXE driver sends a token through PXE configs when booting the deploy ramdisk, and the agent driver could do something similar. More preferable would be to send a secret through some out of band mechanism, and use that secret to authenticate the agent ramdisk to Ironic.

This is TBD and will likely not be implemented in the first iteration of this spec.

Other end user impact

An end user will not interact directly with features provided by this driver.

Scalability impact

This change will involve communication between the agent and Ironic, which may have some impact on performance. However, as the agent is able to directly download and write images, Ironic will no longer have image traffic going through it, and so overall network traffic by Ironic may be less.

Additionally, this driver may allow more nodes to be managed by a single conductor, as the conductor only makes API calls to the agent, rather than writing image data.

Ironics API servers may end up doing more work if the agent is used in a long-running model, as agents will be heartbeating periodically via the API.

The database is updated on each heartbeat, and so may also see extra load in this scenario. However, this update call should be fairly fast, fairly infrequent, and is done in the background in the conductor, so this should cause only a small impact. This can be mitigated easily by scaling the conductor cluster.

Performance Impact

This driver will not change the performance characteristics of any existing code.

The driver does lock nodes sometimes - however, only during deploy and tear_down will the lock be held for a significant amount of time.

The driver should behave similarly to the PXE driver when the hash ring is rebalanced.

Other deployer impact

To use this driver, deployers need to:

- Explicitly enable the driver in the configuration.
- Register nodes with the driver.
- Build an agent image using the tools available in the ironic-python-agent project.

A single configuration option will be added:

- agent.heartbeat_timeout: how long to wait before deciding that an agent is no longer running. Defaults to 300 seconds.

The driver classes added include:

- agent_ssh (AgentDeploy + SSHPower)
- agent_ipmitool (AgentDeploy + IPMIPower)
- agent_pyghmi (AgentDeploy + NativeIPMIPower)
- fake_agent (AgentDeploy + FakePower)

The agent will rely on access to the following services:

- Glance and/or Swift
- Neutron for DHCP

The references section below includes a diagram of an example architecture for running Ironic with the agent driver.

Developer impact

This change should not impact other Ironic developers.

Implementation

Assignee(s)

Primary assignee: JoshNang

Other contributors:

- jroll
- russell_h
- JayF
- dwalleck

Work Items

- Implement the driver.
- Add a diskimage-builder element for IPA.
- Add devstack support for running Ironic with the IPA and PXE drivers *concurrently*. This is crucial for testing.
- Write Tempest tests for Ironic running with the agent driver.

Dependencies

None.

Testing

The plan is to use the existing tempest tests, but with this driver specified. This will require changes to tempest and devstack.

It is critical that deployers can use the PXE and IPA drivers in the same environment. Tempest tests should explicitly test for this support.

Documentation Impact

There will need to be clear documentation on how to run Ironic with the agent driver.

References

- Ironic Python Agent wiki
- Ironic Python Agent repo
- Example agent Architecture

5.15.4 Mechanism to cleanup all ImageCaches

https://blueprints.launchpad.net/ironic/+spec/cleanup-all-imagecaches

This spec talks about creating a mechanism to cleanup all the image caches (leveraging most of the code that is already in pxe driver)

Problem description

The cleanup method for cleaning up ImageCache (or its subclasses) `_cleanup_caches_if_required` is situated inside pxe driver. Hence, any other subclass of ImageCache situated in any other file, cannot use this cleanup method.

This code was introduced in this commit: https://review.openstack.org/#/c/92625

Proposed change

The reason why the cleanup method is placed in pxe driver is because it needs to know other subclasses of ImageCache to do efficient cleaning to make up for the required space in the cache. Other ImageCaches which are situated in the same filesystem are also cleaned up if enough space is not available after cleaning up the current cache. This method hardcodes `InstanceImageCache` and `TFTPImageCache` to be cleaned up for any cache cleanup.

The problem could be solved as follows:

- Each of the subclasses of ImageCache that wishes to be used in cleanup of other caches (currently `InstanceImageCache` and `TFTPImageCache`) add a decorator `@image_cache.cleanup(priority=N)`. N should be a positive integer which signifies the priority in which the cache should be cleaned up when extra free space is required after cleaning up the provided cache. Higher value of N means higher priority. If two caches have same value of N (probably at different places in code), then the order of cleanup of those two caches is not predictable (it will depend on which of the two module is loaded first).

  Currently `InstanceImageCache` being large, should be cleaned up before `TFTPImageCache`. Hence they will have it as below:

```
@image_cache.cleanup(priority=50)
class InstanceImageCache(...)
...


@image_cache.cleanup(priority=25)
class TFTPImageCache(...)
...
```

- There would be list `cache_cleanup_list` in image_cache.py which would contain a sorted list of instances of `ImageCache` to be considered for cleanup. The decorator function `cleanup` adds an instance of the subclass of `ImageCache` into list. This list will be kept sorted in non-increasing order of `priority` after adding the entry.
- The method `_cleanup_caches_if_required` will be moved to ironic.common.image_cache and renamed to `clean_up_caches`. The method `_cleanup_caches_if_required` currently uses a hardcoded list of caches to be cleaned for extra space. Instead of that, the newly proposed method will just use the list `cache_cleanup_list`.
- `PXEImageCache` will not have decorator `@image_cache.cleanup`.

Alternatives

The code to cleanup can continue to exist in the pxe driver and pxe driver can import and maintain the hardcoded list of all the caches across the source tree. This is hard to maintain and not logical.

Another alternative is that the method can be moved to a common place, but the cleanup may be initiated on the caches that the module would know about. This would not be as efficient as the proposed solution, as proposed solution can consider more caches for cleaning up and making up the required space.

Data model impact

None.

REST API impact

None.

Driver API impact

None.

Nova driver impact

None.

Security impact

None.

Other end user impact

None.

Scalability impact

None.

Performance Impact

None.

Other deployer impact

None.

Developer impact

- Developers wishing to introduce a new subclass of `ImageCache` can add the decorator `@image_cache.cleanup` and assign an `priority` in interest of overall ironic.
- The subclass of ImageCache that wishes to be cleaned up should not take any parameters in the constructor.

Implementation

Assignee(s)

rameshg87

Work Items

- Refactor `_cleanup_caches_if_required` from ironic.drivers.modules.pxe to ironic.common.image_cache.
- Modify the unit tests for fetch_images() method in ironic.tests.drivers.test_pxe

Dependencies

None.

Testing

Currently existing unit tests will be modified in accordance with the proposed behavior.

Documentation Impact

None.

References

None.

5.15.5 More robust device status checking with fuser

https://blueprints.launchpad.net/ironic/+spec/device-status-check-with-fuser

Implement a more robust device status checking with fuser to avoid device is busy issues after partitioning.

Problem description

Right after partitioning, we have a sleep(3) call to avoid the device is busy problem. A less error-prone solution is to check with fuser, whether there is any process currently using the disk.

Proposed change

Replace the sleep call with a check of the mounted device with fuser. fuser returns with exit code 0 if at least one access has been found and it lists the processes. In case theres no access, it returns with exit code 1 with no output.

Alternatives

- lsof can also be used for checking for open files. There isnt any advantage or disadvantage in this use-case.

Data model impact

None

REST API impact

None

Driver API impact

None

Nova driver impact

None

Security impact

None

Other end user impact

The deployment can raise InstanceDeployFailure after partitioning if the device is not available after a configurable number of retries.

Scalability impact

None

Performance Impact

Instead of sleeping every time for 3 seconds, fuser will check the status immediately after the partitioning is done. This can result in an shorter deployment time.

Other deployer impact

New config options in the `disk_partitioner` group:

- **check_device_interval: After Ironic has completed creating the partition**
  table, it continues to check for activity on the attached iSCSI device status at this interval prior to copying the image to the node. Default is 1 second.
- **check_device_max_retries: Number of retries for checking the device status.**
  Default is 20.

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
ifarkas

Other contributors:
None

Work Items

- Add config options to `disk_partitioner` group.
- Implement the device status check with LoopingCall.

Dependencies

- This patch requires fuser to be installed.

Testing

- Unit tests

Documentation Impact

Documentation should include instructions on how to configure the device status check.

References

None

5.15.6 DRAC Management driver for Ironic

https://blueprints.launchpad.net/ironic/+spec/drac-management-driver

The proposal presents the work required to add support for boot management features for Dell Remote Access Controllers.

Problem description

Dell Remote Access Controller is an interface card from Dell offering remote system management. This proposal adds the boot management capabilities for DRAC.

Proposed change

- Create a DracManagement class and implement the following methods interacting with the WS-Management API (WS-Man) described in the `DCIM BIOS and Boot Management Profile` using the python binding of the OpenWSMAN library:
  - **set_boot_device() - To set the boot device for a node. The `persistent`** flag will be supported to indicate if the change should be applied for the next boot only, or persistently.
  - **get_boot_device() - To get the current boot device of a node with the** indication whether its persistent, or not.
  - **get_supported_boot_devices() - To get a list of the supported boot** devices of a node. The supported boot devices will be `disk` and `pxe`.
- Add DracPXEDriver class to the list of the available drivers, which uses the PXEDeploy, DracPower, and DracManagement interfaces. (The above change is a prerequisite for this one, because PXE requires setting the boot device to network).

Alternatives

None

Data model impact

None

REST API impact

None

Driver API impact

None

Nova driver impact

None

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

None

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
ifarkas

Other contributors:
None

Work Items

- Add methods supporting boot management to the management interface.
- Create the DracPXEDriver class.

Dependencies

- This feature depends on the python binding of the OpenWSMAN library which was introduced by the power management interface of the DRAC driver.
- This feature requires 11th or 12th generation of Dell PowerEdge servers.

Testing

- Unit tests
- 3rd-party CI: we would like to do it for this driver, but do not have sufficient hardware available at this time.

Documentation Impact

None

References

- Spec for ManagementInterface
- OpenWSMAN library
- DCIM BIOS and Boot Management Profile 1.2

5.15.7 DRAC Power Driver for Ironic

https://blueprints.launchpad.net/ironic/+spec/drac-power-driver

The proposal presents the work required to add support for power management features for Dell Remote Access Controller in Ironic.

Problem description

Dell Remote Access Controller is an interface card from Dell offering a remote system management. This proposal adds the power management capabilities for DRAC.

Proposed change

Adding a new DracDriver to the list of available drivers in Ironic and implementing the DracPower module to interact with WS-Management API (WS-Man) described in the `DCIM Base Server and Physical Asset Profile` using the python binding of the OpenWSMAN library.

Alternatives

There are other ways to interact with WS-Management endpoints but they are wrappers around Open-WSMAN command-line client. These are:

- Recite
- Python WSMAN API

Data model impact

None

REST API impact

None

Driver API impact

None

Nova driver impact

None

Security impact

Admin credentials will be stored unencrypted in the DB and they will be visible in the driver_info field of the node when a node-show is issued.

Other end user impact

The following driver_info fields are required:

- drac_host: hostname or IP of the WS-Man endpoint
- drac_port: port of the WS-Man endpoint (*default value is 443, assuming the user configured the endpoint in secure mode*)

- drac_path: path of the WS-Man endpoint (*default value is /wsman*)
- drac_protocol: protocol of the WS-Man endpoint (*default value is https, assuming the user configured the endpoint in secure mode*)
- drac_username: username for the WS-Man endpoint
- drac_password: password for the WS-Man endpoint

Scalability impact

None

Performance Impact

None

Other deployer impact

None

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
ifarkas

Other contributors:
None

Work Items

- Add DracDriver
- Implement DracPower module for the DracDriver

Dependencies

- This feature depends on the python binding of the OpenWSMAN library. Its released under a simplified BSD licence and is available as a supported package in Ubuntu and Red Hat repositories.
- This feature requires 11th or 12th generation of Dell PowerEdge servers.

Testing

- Unit tests
- 3rd-party CI: we would like to do it for this driver, but do not have sufficient hardware available at this time.

Documentation Impact

The required driver_info properties need be included in the documentation to instruct operators how to use Ironic with DRAC.

References

- OpenWSMAN library
- DCIM Base Server and Physical Asset Profile 1.0

5.15.8 Enabling IPMI double bridge support

https://blueprints.launchpad.net/ironic/+spec/enabling-ipmi-double-bridge-support

This blueprint proposes ipmi double bridging support in ironic.

Problem description

Currently, ironic IPMI driver(ipmitool) does not support bridging.

Many of the recent server architecture is based on distributed management. For a chassis that has n number of servers, the management is delegated from the core controller to many satellite controllers, which mandates the need for bridging.

Proposed change

- When registering an baremetal node which requires bridging, the appropriate parameters should be specified to ironic IPMI power driver as follows:
  - -i ipmi_bridging=<single/dual/no>
  - -i ipmi_local_address=<VALUE>
  - -i ipmi_transit_local_address=<VALUE>
  - -i ipmi_transit_channel=<VALUE>
  - -i ipmi_transit_address=<VALUE>
  - -i ipmi_target_channel=<VALUE>
  - -i ipmi_target_address=<VALUE>

    The parameters can be specified based on the hardware being registered. i.e, In order to perform an double-bridge, an user can just specify transit_address and target_address, rest is taken care by ipmi. But some hardware will mandate to specify transit_channel and target_channel, if they are using different channels.

    The parameter ipmi_bridging should specify the type of bridging required(single/dual) to access the baremetal node. If the parameter is not specified, the default value will be set to no

    Single Bridging:

    ironic node-create -d pxe_ipmitool [-i ipmi_local_address=VALUE] <-i ipmi_bridging=single> <-i ipmi_target_channel=VALUE> <-i ipmi_target_address=VALUE>

    The parameter ipmi_local_address is optional. If the parameter is not specified, it is auto discovered by ipmitool

Double Bridging:

ironic node-create -d pxe_ipmitool [-i ipmi_local_address=VALUE] [-i ipmi_transit_local_address=VALUE] <-i ipmi_bridging=dual> <-i ipmi_transit_channel=VALUE> <-i ipmi_transit_address=VALUE> <-i ipmi_target_channel=VALUE> <-i ipmi_target_address=VALUE>

The parameters ipmi_local_address and ipmi_transit_local_address are optional. If the parameters are not specified, it is auto discovered by ipmitool

- Ironic IPMI driver should be modified to parse the above information and perform ipmi operations with appropriate parameters.

Alternatives

None

Data model impact

None

REST API impact

None

Driver API impact

None

Nova driver impact

None

Security impact

None

Other end user impact

None

Scalability impact

Depends on the number of parallel IPMI sessions that can be supported by the underlying BMC. When the sessions are exhausted, IPMI retry option can be used to get the handle of a session.

Performance Impact

If the underlying BMC is designed for federated management, where there might be one main controller and many sub controllers, there will not be any impact. However if only one controller exists in the BMC that manages all the nodes, then the sessions might be slower when it reaches its threshold.

Other deployer impact

When an node which mandates bridging is being registered, provide the appropriate parameters:

- -i ipmi_bridging=<single/dual/no>
- -i ipmi_local_address=<VALUE>
- -i ipmi_transit_local_address=<VALUE>
- -i ipmi_transit_channel=<VALUE>
- -i ipmi_transit_address=<VALUE>
- -i ipmi_target_channel=<VALUE>
- -i ipmi_target_address=<VALUE>

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
rh-s

Other contributors:
bmahalakshmi

Work Items

- Include functionality to IPMI driver(ipmitool) to check if the underlying ipmitool utility supports bridging.
- Changes to IPMI driver to parse the bridging parameters.
- When a node being provisioned has bridging configuration specified, perform all ipmi operations with appropriate parameters.

Dependencies

IPMITOOL_1_8_12

Testing

Unit test cases to test IPMI driver with bridging enabled and disabled

Documentation Impact

Documentation should reflect the parameters that can be provided during registering an node to enable bridging operation.

References

- http://manpages.ubuntu.com/manpages/trusty/man1/ipmitool.1.html
- http://sourceforge.net/p/ipmitool/mailman/ipmitool-cvs/?viewmonth=201001

5.15.9 API to Get driver_info Properties

https://blueprints.launchpad.net/ironic/+spec/get-required-driver-info

This blueprint proposes an API that returns all the driver_info properties, along with a description for each property.

Problem description

It is possible to create a node without specifying any driver_info properties in the initial POST this is reasonable and fine. However, the API does not expose the list of driver_info properties, nor which are required for the nodes driver. The client cannot know what fields/properties to send in subsequent PATCH requests without reading Ironics developer docs (or source code!).

To address the above, an API is proposed, that returns the driver_info properties, along with a description for each property. The description will include whether the property is required or not.

Being an API, it may be consumed by humans and applications.

Proposed change

RESTful web API

The RESTful web API will be enhanced with:

GET /v1/drivers/<driver>/properties

where <driver> is the name of the driver.

If unsuccessful, eg an invalid driver name was specified, it returns HTTP status 404 and an error message.

If successful, it returns HTTP status 200 and the response (in Json format) is a list of

```
<property name>: <property description>
```

where <property description> is a description of the property, including whether it is required or optional or any other special circumstances.

Eg: `GET /v1/drivers/pxe_ssh/properties` might return:

```
{"pxe_deploy_ramdisk": "UUID... Required.",
 "ssh_address": "IP address or hostname of the node to ssh into. Required.",
 "ssh_virt_type": "virtualization software... Required.",
 "ssh_user_name": "username to authenticate as. Required.",
 "ssh_key_contents": "private key(s). One of this, ssh_key_filename,
      or ssh_password must be specified."
 "ssh_key_filename": "filename ... One of this, ssh_key_contents,
      or ssh_password must be specified."
 "ssh_password": "password... One of this, ssh_key_contents, or
      ssh_key_filename must be specified."
 "pxe_deploy_kernel": "UUID... Required."}
}
```

CLI subcommand

The `driver-properties` subcommand will be added:

```
ironic driver-properties <driver_name>
```

It returns a table with the driver_info properties of the specified driver. For each property, this information is displayed:

- name of the driver_info property
- description

Eg:

```
$ ironic driver-properties fake_ipminative
+---------------+---------------------------------+
| Property      | Description                     |
+---------------+---------+-----------------------+
| ipmi_address  | IP of the node's BMC. Required. |
| ipmi_password | IPMI password. Required.        |
| ipmi_username | IPMI username. Required.        |
+---------------+---------------------------------+
```

For invalid driver names, it returns:

```
The driver '<invalid-driver-name>' is unknown. (HTTP 404)
```

Required vs optional properties

The driver properties are specific to each driver, and depend on the interfaces (power, deploy, console, rescue, management) of a driver. It is at the interface implementation where we identify which properties are required and which are optional. Having said that, it isnt all black and white. For example, a driver with the `SHPower power` interface requires one of `ssh_key_contents`, `ssh_key_filename`, or `ssh_password` properties to be specified. Handling this exactly one of these must be specified case might be reasonable, but what happens if there is a desire for one or more of these must be specified, exactly X of these must be specified, B must be specified if A is specified, or if A is specified, B or C must be specified?

After *discussing this*, we decided to take the approach of indicating, as part of the description, whether a property is required or not along with any constraints on that. No explicit required field will be returned.

How the API service gets the information

A conductor service can handle one or more different drivers. There could be different versions of conductor services running, different versions of the API service running, and different versions of drivers available via the different conductor services. Ironic currently has a mechanism for versioning the conductor service and api service (via the RPC_API_VERSION). However, there is no mechanism yet for versioning of drivers.

When/if Ironic has a mechanism for versioning the drivers, the API (and code) can be updated to use the driver version to get the driver properties specific to that version.

Driver upgrades (resulting in one or more conductor services being restarted) need to be considered, since driver upgrades could include changes to their properties. There will be upgrade windows, during

which different conductor services may be handling different drivers due to an upgrade. This specification assumes that the upgrade window is small, and that after an upgrade, all conductor services will be handling the same driver versions. (The right solution is to have explicit driver versioning; an intermediate solution might be to allow the user to explicitly specify a conductor service when querying for driver information, but that doesnt seem to be the right approach to take.)

After the conductor services are upgraded, all the API services should be restarted. So during the upgrade window, the API services may return incorrect/different driver property information, but after the upgrade is done, the information should be correct again.

Although an API service could access/instantiate the drivers directly, that would only give the service access to local drivers. These drivers may not be the actual drivers that the conductor services use. Furthermore, since the drivers talk to hardware, the API service shouldnt be allowed to access them directly.

The conductor service, then, is the gateway to getting driver properties. Two approaches were considered:

1. the API service queries, via RPC, a conductor service, to get the driver properties. It picks the first conductor service (any one will do if we assume that all the conductors are handling the same version of the driver). **This is the approach we will take.**
2. the API service queries the DB to get driver properties that the conductor services have placed there. When a conductor service starts, it adds the property information for each of the drivers it can handle, to a DB table. Since more than one conductor could be handling the same driver, the driver information would be added to a new DB table, different from the conductor table.

For both of these approaches, making an RPC or DB call for each user request may become a performance issue; especially if the user requests are generated by some automated system. Since the information is static for the lifetime of the conductor services (or longer), it makes sense for the API services to cache the information locally.

If an upgrade (where a conductor driver is updated) occurs, all the API services must be restarted after the conductor service upgrades are completed. This will clear out the caches, to make sure that the API services get the most recent drivers information.

A cache-refresh mechanism could be added, but the information is relatively static and only changes when a driver changes. Driver changes should be infrequent enough that having the API services restarted after conductor services are upgraded should suffice.

Since there doesnt seem to be much gain with storing the driver information in the database since caching will be done, having the API service query a conductor service for the driver information (approach #1) will be implemented.

Alternatives

The driver_info information could be made available in a non-API fashion:

- document the information.
  - pros: no code changes at all, no need to write this specification
  - cons: user needs to know where to find the documentation; documentation needs to be kept up-to-date; more difficult to write automation tools to extract this information
- read the code.
  - pros: no additional code changes required; no need to write this spec; will always be *the source of truth*

- cons: very user-unfriendly; user needs to know python and know where to find the appropriate code.

Given that we think having an API is a GOOD THING, these approaches were ruled out.

This doesnt describe alternative RESTful web API, CLI commands or response outputs, because the proposed API is consistent with the existing API, but clearly there are alternatives.

Data model impact

This will add an internal cache to each API service. The database is not affected.

REST API impact

See *RESTful web API* section above for a description of the new request.

Driver API impact

All the driver interfaces (DeployInterface, PowerInterface, ConsoleInterface, RescueInterface, VendorInterface, ManagementInterface) will/must have a new method:

```
@abc.abstractmethod
def get_properties(self):
    """Return the properties of the interface.

    :returns: a dictionary with <property name>:<property description>
              entries
    """
```

Nova driver impact

None

Security impact

None

Other end user impact

See *CLI subcommand* section above for the CLI subcommand.

Scalability impact

None

Performance Impact

Negligible.

Other deployer impact

Requirement that all the API services must be restarted after an upgrade of the conductor services.

Developer impact

None except for doing reviews. Well, making sure the list of properties is updated in the code.

Implementation

Assignee(s)

Primary assignee:
rloo

Other contributors:
None

Work Items

Bug:

- API does not expose required driver_info (https://bugs.launchpad.net/ironic/+bug/1261915)

Patches:

- Implement API to get driver properties (https://review.openstack.org/#/c/73005/)
- Add driver-properties command (https://review.openstack.org/#/c/76338/)

Dependencies

None

Testing

Since the information is static, Ironic unit tests are sufficient.

Tempest testing should be added if the QA team feels it is in the best interest of Tempest to check the output of common drivers.

Documentation Impact

The CLI subcommand will need to be documented, but the docs team have a script that generates the documentation via issuing ironic commands.

One or more guides (operators and/or deployment) will need to mention that all the API services need to be restarted after an upgrade of all conductor services.

References

discussion about how to handle required vs optional properties. Starting from 2014-07-08T14:18:06: http://eavesdrop.openstack.org/irclogs/%23openstack-ironic/%23openstack-ironic.2014-07-08.log

5.15.10 iLO IPA Deploy Driver

https://blueprints.launchpad.net/ironic/+spec/ilo-virtualmedia-ipa

Add ability to provision proliant baremetal nodes (having iLO4 and beyond) by booting the baremetal node with virtual media and using IPA to deploy the image.

Problem description

IPA project provides a more powerful ramdisk for doing deploy from the conductor node. But IPA has the following issues:

- Some customers dont prefer PXE protocol in their environment because of the following issues:
  - PXE uses TFTP to transfer the files which is unreliable because it uses UDP.
  - PXE is not suited for some network topologies where the relaying of PXE requests might be required in routers to enable PXE working for the whole network.
- Deployers require an extra tftp service to be running on the conductor node.
- Currently the admin token required to call nodes vendorpassthru cannot be transmitted securely to the baremetal node.

Proposed change

The below review introduced a new mechanism for booting proliant machines with virtual media. http://specs.openstack.org/openstack/ironic-specs/specs/juno/ironic-ilo-virtualmedia-driver.html

The methods `setup_virtual_media_boot` introduced in the above review can be used to boot up a baremetal node with the deploy ISO image. A new class `IloVirtualMediaAgentDeploy` can be added which will setup the machine to be booted with virtual media instead of PXE.

The vendor interface `AgentVendorInterface` can be reused to continue the deploy and complete it.

This change will also enable the admin token to be handed off securely to the baremetal node through OOB channel over virtual media.

Alternatives

The proliant machines can continue to work booting the agent ramdisk with PXE.

Data model impact

One new parameter `deploy_iso` will be used in driver_info to boot up the node for deploy. `deploy_iso` will contain the glance UUID of bootable ISO built with agent ramdisk.

REST API impact

None.

Driver API impact

None.

Nova driver impact

None.

Security impact

None.

Other end user impact

None.

Scalability impact

None.

Performance Impact

None.

Other deployer impact

This method of deploy no longer requires an extra service (like tftp service in case of pxe driver) to be running on the conductor node.

Developer impact

None.

Implementation

Assignee(s)

Primary assignee:
rameshg87

Work Items

- Add `IloVirtualMediaAgentDeploy` which implements base.DeployInterface.

Dependencies

None.

Testing

Unit tests will be added for all the code. Tempest tests for this will be considered later.

Documentation Impact

The procedure for configuring the proliant baremetal node will need to be documented. This will be documented in rst format in doc/ directory in ironic source tree. The contents of this file can be put in ironic wiki as well.

References

None.

5.15.11 iPXE boot

https://blueprints.launchpad.net/ironic/+spec/ipxe-boot

This blueprint presents the work needed to add support for iPXE in Ironic.

Problem description

As the size of our deploy ramdisk would continue to increase (Ironic Python Agent) we need a more reliable way to transfer such data via the network without relying on TFTP. The problem with TFTP is that its unreliable and any transmission error will result consequently in boot problems (The first T in TFTP stands for trivial).

Proposed change

By adding support for iPXE we would have the ability to transfer data through HTTP which is a reliable protocol.

- **New config options:**
  - ipxe_enabled: Whether iPXE is enabled or not.
  - ipxe_boot_script: The path to the main iPXE script file.
  - http_server: The IP address of the HTTP server.
  - http_root: The HTTP root path.
- When generating the PXE configuration file the kernel and initrd parameters should contain the HTTP URL for the files and not the TFTP path.
- All the configuration files, ramdisks and kernels will now be put in the HTTP directory instead of the TFTP directory.
- The pxe_bootfile_name config option should point to the iPXE image (undionly.kpxe).
- A configuration template for iPXE.
- The pxe_config_template config option should point to the iPXE configuration template.
- An iPXE script file (ipxe_boot_script config option) which is the file fetched by the client after it has loaded the iPXE image, and from there the script will load the MAC-specific iPXE configuration file for that request.
- When passing the DHCP boot options to Neutron we also have to pass the HTTP link pointing to the iPXE script file.

Its important to note that Ironic is not responsible for managing the HTTP server, just like the TFTP server, it should be configured and running on the Node that ironic-conductor was deployed.

Another important note is that the iPXE image (undionly.kpxe) used for chainloading is sent to the clients via TFTP, so we still need a TFTP server up and running, this is the only TFTP transaction in the whole process, once the client has loaded iPXE, everything happens over HTTP.

Alternatives

Continue to use the standard PXE and rely on the TFTP protocol to transfer the data.

Data model impact

None

REST API impact

None

Driver API impact

None

Nova driver impact

None

Security impact

While not part of work proposed by this spec, iPXE supports using the HTTPS protocol which allows encrypting all communication with the HTTP server, this patch can be considered a plumbing work for that to be implemented in the future.

Other end user impact

To enable iPXE users would have to set the http_root, http_server and ipxe_enabled configuration options along with the tftp_root and tftp_server options.

Scalability impact

As a future work, we can add support to be able to fetch images and configuration files directly from Glance or Swift since those are already scalable.

Performance Impact

TFTP can be extremely slow, so fetching data over HTTP can improve the speed of transferring the images from the conductor to the Node being booted.

Other deployer impact

New config options:

- ipxe_enabled: Whether iPXE is enabled or not.
- ipxe_boot_script: The path to the main iPXE script file.
- http_server: The IP address of the HTTP server.
- http_root: The HTTP root path.

By default iPXE will be disabled and so should not change anything on the current flow to deploy/configure Ironic. In the future since we are moving towards having the Ironic Python Agent to be the standard provisioning method, we might want to enable iPXE by default as part of that effort.

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
lucasagomes

Other contributors:
None

Work Items

See the Proposed change section.

Dependencies

A HTTP server up and running.

Testing

- Unit tests.
- Add support to DevStack to be able to configure Ironic to use iPXE.

Documentation Impact

Documentation should be modified to instruct operators about how to enable and configure Ironic to use iPXE.

References

None

5.15.12 iLO Power Driver for Ironic

https://blueprints.launchpad.net/ironic/+spec/ironic-ilo-power-driver

This proposal adds the ability to manage power control for HP ProLiant servers via iLO using iLO client python library.

Problem description

The HP iLO subsystem is a standard component of HP ProLiant servers that simplifies initial server setup, server health monitoring, power and thermal optimization, and remote server administration. Our proposed Ironic IloDriver will expose these capabilities in OpenStack to manage HP Proliant servers.

This proposal covers support for power management interfaces using iLO.

Proposed change

To add a new IloPower() module which will conform to base.PowerInterface. This module uses iLO credentials (ilo_address, ilo_username, ilo_password), specified in driver_info property of the node to connect to target iLO. It makes use of iLO client in proliantUtils library to talk to iLO.

HP iLO also supports more advanced power management features for monitoring and power capping. We would like to add support for these advanced vendor specific features later on.

NOTE: Even though iLO uses SSL over port 443 for communication, ssh key based authentication is not supported for RIBCL communication. iLO username/password will need to be provided for talking to iLO.

Alternatives

IPMI standard specification can be used for the power management. But adding a new power module allows the solution to be consistent with other future iLO drivers for deploy, management in using a single iLO interface to do the operations.

There is no functional benefit as of now in choosing this module over IPMI.

Data model impact

None

REST API impact

None

Driver API impact

None

Nova driver impact

None

Security impact

iLO admin credentials will be stored unencrypted in the Ironic DB. This will also be visible with the driver_info of the node when a node-show is issued. But only the ironic admin user will have access to the Ironic DB and node details.

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

The following driver_info fields are required:

- `ilo_address` - hostname or IP address of the iLO.
- `ilo_username` - the username for the iLO with administrator privileges.
- `ilo_password` - the password for `ilo_username`
- `ilo_client_timeout` - the timeout for iLO operations. The default value will be 60 seconds.
- `ilo_client_port` - the port to be used by iLO client for iLO operations. The default value will be 443.

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
Ramakrishnan G (rameshg87)

Other contributors:
Anusha Ramineni (anusha_08)

Work Items

Implement a new power module, IloPower, conforming to base.PowerInterface.

Dependencies

- This feature is targeted for HP ProLiant servers with iLO4 and above. This power module might work with older version of iLO (like iLO3), but this will not be officially tested by the iLO driver team.
- Depends on proliantutils library.

Testing

Unit tests will be added, mocking proliantutils library.

Tempest tests will be considered later when more advanced module of IloDriver like deploy are available in the ironic tree.

Documentation Impact

The required driver_info properties need be included in the documentation to instruct operators how to use iLO Driver with Ironic.

References

proliantutils library: https://github.com/hpproliant/proliantutils https://pypi.python.org/pypi/proliantutils

HP iLO4 User Guide: http://h20628.www2.hp.com/km-ext/kmcsdirect/emr_na-c03334051-10.pdf

HP Power Capping and HP Dynamic Power Capping http://bit.ly/1m8sbEi

5.15.13 iLO Virtual Media iSCSI Deploy Driver

https://blueprints.launchpad.net/ironic/+spec/ironic-ilo-virtualmedia-driver

Add ability to provision proliant baremetal nodes (having iLO4 and beyond) by booting the baremetal node with virtual media and using iscsi from conductor node to deploy the image (reusing existing deploy mechanism).

Problem description

- Today Ironics PXE reference driver uses pxe protocol to boot the machine. Some customers dont prefer PXE protocol in their environment because of it unreliability and security issues.
- Today Ironics PXE reference driver passes the keystone authentication token in clear text over tftp on the data network to the baremetal node.

Proposed change

The proposed change for Ironic deploy will happen in two stages:

- Refactor the iSCSI deploy code in current pxe deploy driver into a new module ironic/drivers/modules/iscsi_deploy.py so that it can be reused in a new deploy driver.
- Add two new methods `create_vfat_image` and `create_iso_image` in ironic/common/images.py for creating vfat images and iso images respectively. The vfat images will be used for passing the token and parameters to the ramdisk when it is booted over virtual media. The ISO image will be used for booting up the kernel/ramdisk on the baremetal machine.
- Add a new module ironic/common/swift.py to manage objects in swift.
- Add two generic methods `setup_virtual_media_boot` and `cleanup_virtual_media__boot` which helps in setting up and cleanup up virtual media for booting respectively.
- Create a new deploy module named `IloVirtualMediaIscsiDeploy` in ironic/drivers/modules/ilo/deploy.py which adheres to `base.DeployInterface`.
- Create a new class `VendorPassthru` which adheres to `base.VendorInterface` in ironic/drivers/modules/ilo/deploy.py. Implement a vendor passthru method `pass_deploy_info` in it.
- The reboot() method in `IloPower` module will be changed.

Changes in Detail

Virtual media for booting

This class exposes the following methods:

setup_virtual_media_boot()

- Validate that the nodes iLO has virtual media feature enabled using proliantutils module. If the node doesnt have virtual media feature, it comes out with error.
- If `boot_parameters` is not empty:
  - Create a virtual floppy image containing the user token and a config file, which contains `boot_parameters`.
  - Upload the virtual floppy image to swift. Set `X-Delete-After` for swift to delete the image after `deploy_helper_images_ttl` minutes. The default value will be 10 minutes. Upload to container `swift_ilo_container` whose default value will be ironic_ilo_container.
  - Generate the swift temp url for virtual floppy image. Let it be named as floppy_image_temp_url. The timeout of tmpurl will be `deploy_helper_images_ttl` minutes.
  - Attach floppy_image_temp_url as virtual media floppy in the iLO. Set the timeout to `deploy_helper_images_ttl`
- Generate tmpurl for `boot_iso` and attach it as virtual media cdrom. The timeout of tmpurl will be `deploy_helper_images_ttl` minutes.
- Set the baremetal node to boot from virtual media cdrom for the next boot using proliantutils module with BOOT_ONCE option.

cleanup_virtual_media_boot()

- Remove the virtual media floppy image uploaded to swift for the node. The object name in swift will be image-<node uuid>

ironic/common/swift.py

This module will provide a class SwiftAPI which will handle the creation and managing swift objects. This SwiftAPI module by default will use admin credentials for talking to swift. The user of this module may also choose to pass `user`, `tenant_name`, `key`, `authurl` to create the swiftAPI object.

- *upload_object* - Creates the container if required and requested, and then creates the new object in swift using swiftclient. Returns the swift object id.
- *delete_object* - Deletes the object from the mentioned swift container.
- *get_tmp_url* - This will call the `swift_utils.generate_temp_url()` which is available in python-swiftclient.

ironic/common/images.py

Add a new method `create_vfat_image` which helps in creating virtual floppy images. This method takes the files and parameters to be included in the floppy image as input, and then creates a vfat floppy image.

Add a new method `create_iso_image` which helpsin creating ISO images. This method takes the files and parameters to be included in the ISO as input, and then creates the ISO image.

The common components between the two methods above will be reused.

ironic/drivers/modules/iscsi_deploy.py

This module will refactor every method belonging to the iscsi deploy mechanism from the pxe driver. The following methods will be moved to the new module:

- parse_instance_info()
- _cache_instance_image
- InstanceImageCache
- _check_image_size
- _destroy_images()
- _get_deploy_info
- _continue_deploy()

Minor changes will be required in the refactoring to remove the pxe portions out of the above methods.

IloVirtualMediaIscsiDeploy

This class will implement the following:

- *validate()* - Validates that node has ports added, parses deploy_info(), checks that conductor api url is available, and validates that `deploy_iso` property exists in driver_info of node. Most of the functions from refactored iscsi_deploy is used.
- *deploy()* - Caches instance image, uses virtual media boot helper method `setup_virtual_media_boot` to setup the machine for booting with driver_info[deploy_iso].
- *tear_down()* - Powers down the node.
- *clean_up()* - Destroys the images, calls `cleanup_virtual_media_boot` method to clean the temporary floppy images. Decrements the `usage_count` for `boot_iso` in swift and destroys the boot ISO image if the `usage_count` meta-property becomes 0 (if `boot_iso` was created by ironic).

prepare() and take_over() will be empty.

VendorPassthru

Implement a new vendor passthru method pass_deploy_info. This vendor passthru method will call continue_deploy() from iscsi_deploy.py.

After deploying over iSCSI, it checks the following in that order to pick up a boot ISO:

- If user has specified a boot ISO in glance image, then it picks up this.

- Checks if a boot_iso is already available for the mentioned (image, kernel, ramdisk) for the image in swift on `swift_ilo_container` (by hashing the UUIDs of the image, kernel, ramdisk to get a unique name). If the boot iso exists, then the `usage_count` swift meta-property for the swift object is incremented by 1.
- If we still cant find boot ISO, it creates a bootable ISO image, uploads it to swift on `swift_ilo_container` with the generated name and sets `usage_count` to 1.

It then records the information about `boot_iso` in nodes `instance_info[boot_iso]`

IloPower reboot()

If node has `boot_iso` in its instance_info, use `setup_virtual_media_boot` to set the machine to boot from `boot_iso`.

Alternatives

The proliant baremetal machines could be booted with proposed iPXE, but even that will involve booting the machine with PXE to load the iPXE software. Also it would not solve the security issues in token handoff to baremetal node.

Data model impact

The new deploy driver will use two new parameters:

- driver_info[deploy_iso] - This will be used to boot up the node before the deploy.
- instance_info[boot_iso] - This is set by the deploy driver once the baremetal node deploy completes.

REST API impact

One vendor_passthru method will be added:

pass_deploy_info:

- Description: The deploy ramdisk built using deploy-ironic element of diskimage-builder will call this method on the node. It will also pass the required information for completing the deploy after connecting to the baremetal nodes local disk using iSCSI.
- Method type: POST
- Normal response code: 200
- Expected errors: 400: Insufficient/Invalid data sent or some data for deployment missing.
- URL: /{api_version}/nodes/<node-uuid>/vendor_passthru/pass_deploy_info
- Parameters:
  - `address` - Address of the baremetal node.
  - `key` - The deployment key generated by ironic.
  - `iqn` - The iqn of the target disk on baremetal node where the image has to be deployed.
  - `error` - The error message if some error was encountered.
- Body JSON schema:

```
{
 "address": "10.10.1.150"
 "iqn": "iqn-12345678-1234-1234-1234-1234567890abcxyz"
 "key": "1234567890"
 "error": ""
}
```

- Response JSON: None

Driver API impact

None.

Nova driver impact

No changes are required on the nova ironic virt driver. The new iLO driver will continue to use the below 5 parameters set by nova ironic virt driver in the nodes instance_info:

- `image_source`
- `root_gb`
- `swap_mb`
- `ephemeral_gb`
- `ephemeral_format`

Security impact

- The PXE driver requires the admin token to be available on tftp which can be accessed by anyone in the deploy network (since the filename of the token is predicatable, which is token-<node uuid>). In virtual media boot, the user token is sent to the conductor node securely over https through OOB channel. Hence, this deploy method can be used for more secure deployments.
- The virtual floppy image is uploaded to a swift container with user token and is destroyed automatically by swift after the timeout. It is recommended to use a separate container to secure the floppy images.
- Glance backed by swift can be configured to store the images such that the owner of the image and a defined list of admin accounts will be able to access the image. For more information refer using `swift_store_multi_tenant` in [1].

Other end user impact

None

Scalability impact

None.

Performance Impact

None.

Other deployer impact

The cloud operator is supposed to do the following as part of configuring the iLO driver:

- Upload the deploy_iso to glance and mention its UUID in driver_info[deploy_iso].

Also, the user/operator may also optionally specify a `boot_iso` from which the kernel/ramdisk can be booted off for a deploy image. This may be specified as a glance meta-property `boot_iso` for the image to be deployed.

Utilities will be provided in diskimage-builder for creating the deploy ISO.

This method of deploy doesnt require an extra service (like tftp service in case of pxe driver) to be running on the conductor node.

Developer impact

None.

Known Limitation

- If the user needs to reboot the baremetal node, then the reboot needs to be triggered from Ironic (or from Nova).
- If the user needs to issue an inband reboot of the baremetal node (reboot from within the baremetal node), then the baremetal node will fail to boot. In such a case, the user may just issue a reboot from ironic again to get the node booted up.

Implementation

Assignee(s)

Primary assignee:
rameshg87

Work Items

The work will be split up into following separate items (or patches):

1. Refactor the iSCSI deploy code in current pxe deploy driver.
2. Implement the changes to ironic/common/images.py module.
3. Implement the ironic/common/swift.py module.
4. Implement the virtual media boot helper methods, add the new deploy driver and new vendor passthru module.
5. Implement the changes to reboot() method in IloPower.

Dependencies

Depends on hpproliant module:

- https://github.com/hpproliant/proliantutils

Testing

Unit tests will be added for all the code.

Tempest tests for the deploy will be considered later.

Documentation Impact

The procedure for configuring the proliant baremetal node will need to be documented. This will be documented in rst format in doc/ directory in ironic source tree. The contents of this file can be put in ironic wiki as well.

References

1. http://docs.openstack.org/admin-guide-cloud/objectstorage_tenant_specific_image_storage.html

5.15.14 Power driver for SNMP-enabled smart PDUs

Launchpad blueprint:

https://blueprints.launchpad.net/ironic/+spec/ironic-snmp-power-driver

This blueprint proposes a mechanism for remote control of node power by enabling or disabling sockets on a rack power strip. The result will be a much wider generalisation of the hardware that can be controlled by Ironic.

Problem description

Currently Ironics physical hardware support is restricted to servers implementing power control with embedded management hardware, for example a BMC or other system that implements IPMI. This blueprint proposes widening Ironics capabilities by adding support for a class of power devices that are controllable by SNMP, including smart PDUs with network connectivity.

Implementing an interface to smart PDUs would provide the ability to control bare metal compute nodes built from commodity hardware that does not include a BMC. A cost-conscious user may want to use bare metal compute without the additional expense of server equipment that supports integrated power management.

Proposed change

The proposed design would introduce a new driver to Ironic, snmp.py. A driver class would provide the SNMP manager entity which can convey power management operations over SNMP all the way to the SNMP agent running at the PDU. Classes would derive from this base driver to add specific Object Identifiers (OIDs) and methods for interfacing with different vendor equipment.

The proposed design would use the PySNMP package which implements many aspects of the SNMP technology. The PySNMP package supports all SNMP versions e.g. v1, v2c and v3. Smart PDUs from APC appear to support SNMP v1 and v3. The ability to specify a protocol version for each managed PDU would be desirable. By default, version 3 should be used due to its superior security.

Note that this blueprint only proposes power management for baremetal compute node instances.

Note that this blueprint only proposes support for bare metal compute nodes powered by a single outlet. It is assumed that servers with redundant PSUs will also include embedded management such as a BMC.

There does not appear to be a standard MIB for PDU control. Each vendor publishes its own enterprise MIB for power management and monitoring. Conventionally the same enterprise MIB is implemented by all products from a vendor. Addition of a new derived class to support a new vendor MIB requires defining a function to convert outlet number into an SNMP OID, and defining the values to write to turn the outlet on and off. The design should enable the creation more complex interactions if a vendor MIB was defined in a way that required it.

Market research reports indicate the PDU market segmentation is between the following companies (in descending order of market share):

- APC Corp.
- CyberPower Systems Inc.
- Eaton Corp.
- Emerson Network Power
- Raritan Inc.
- Server Technology Inc.
- AFCO Systems
- Enlogic Systems LLC
- Geist Ltd.
- Hewlett-Packard Co.
- Racktivity NV.

In its first implementation the SNMP power driver will support at least the 3 most dominant vendors - APC, CyberPower and Eaton/Pulizzi.

Alternatives

There is no clear alternative mechanism for interfacing with smart PDUs. Screen-scraping of human-formatted data from CLI terminals or web interfaces sounds like a bad idea.

The advantages of an SNMP-based approach are:

- MIBs tend to be a common interface implemented by all products from a vendor. A MIB interface is not susceptible to variations between products.
- Once published a MIB interface is not changed in a backward-incompatible way. A MIB interface is not susceptible to variations between firmware versions.
- Conventionally vendor MIBs are published and freely available.

Options exist for turning symbolic representation of MIB objects into a MIB-independent OID form.

- The Net-SNMP package comes with the *snmptranslate* command-line tool which can turn any MIB object into OID.
- libsmi includes a tool called smidump can be used to convert MIB definitions into Python dictionaries with some hierarchical structure.

- The PySMI pure-Python package is able to parse MIB files into either JSON document or a Python module which PySNMP can readily consume. The PySMI package comes with the *mibdump.py* tool which can be used at the command line for MIB conversion similar to what *snmptranslate* does.
- Current PySNMP has PySMI as a dependency so PySNMP would invoke PySMI automatically to parse a MIB whenever needed.

Note that these approaches are heavyweight solutions. For example, parsing just the APC vendor MIB involves the creation of a hierarchical structure of 2671 dictionaries to represent the OIDs. Only one OID is actually required.

The proposed solution is to manually extract only the required OID definition from the autogenerated output. This works, because a symbolic representation for the OID is produced. The manual transfer is done once: published MIBs are immutable, and the address and semantics of an OID can never change. The result is lightweight, without loss of functionality.

Data model impact

On creation of an Ironic bare metal node, additional attributes would be attached to the node object:

- snmp_driver - The class of power driver to interface with. This will identify a vendor-specific MIB interface to use.
- snmp_protocol - The SNMP protocol version to use: v1, v2c, or v3
- snmp_address - The hostname or IP address of the SNMP agent running at the PDU.
- snmp_community - The write SNMP community name shared between SNMP manager (e.g. Ironic SNMP driver) and SNMP agent (at PDU).
- snmp_outlet - The power outlet number on the power device.

These attributes would be passed through with other instance data to the Ironic SNMP driver and used to generate the SNMP management operations to achieve the required power action.

For full SNMPv3 support additional attributes might need to be added to the node object.

REST API impact

None.

Driver API impact

This driver would implement a complete interface for a power driver. The power driver functionality is orthogonal to the deployment or boot device management, and these interfaces would not be implemented. A new Ironic driver class, derived from base.BaseDriver, will be implemented to couple the existing PXE boot device configuration and deployment driver with the new SNMP power driver.

Nova driver impact

None.

Security impact

Providing access to power management has obvious implications, but these are not substantially different between one mechanism and another. An argument could be made that the PDU outlets provide access to more devices than might otherwise be reachable from Ironic.

If a user was able to effect a change in the attributes associated with her nodes, it could be possible to affect the power of other devices in the system. This is no different from other power mechanisms.

Using SNMP protocol version 3 increases security through use of encryption. SNMP v3 also adds the potential to increase security through options for authentication. This would provide security above the level of other power drivers, but would require management of authentication credentials by Ironic. Support for power driver authentication is not proposed as part of this initial spec.

Other end user impact

Providing remote control of the outlets on a smart PDU creates a dependency on the connection of the power leads attached to the smart PDU. To use the outlets for power control, the mapping between bare metal node and power outlet must be accurately maintained. However, this is no different from any other scenario in which smart PDUs are deployed.

Scalability impact

The scalability load is no different from other mechanisms using a network protocol for power control.

Performance Impact

None.

Other deployer impact

This driver would not be enabled in a default configuration.

To enable this driver in a deployment, driver-specific data would need to be supplied as bare metal node properties. The mapping of power outlets to bare metal nodes would also need to be determined.

Developer impact

There should be no impact on other Ironic development activity.

Implementation

Assignee(s)

Primary assignee:
<stigtelfer>

Assistance from other contributors would be welcome.

Work Items

- Develop the framework and base class SNMP power driver.
- Add derived classes for interfacing with various PDU vendor MIBs.
- Investigate the feasibility of implementing third party CI for PDU hardware.

- Investigate the feasibility of implementing a virtualized PDU for Tempest.

Dependencies

This project would have a dependency on the PySNMP module. The dependency could be relaxed to a dynamic runtime dependency that only applied if the configuration was enabled. This would also enable unit testing without importing PySNMP.

Testing

The standard driver unit tests can easily be ported to apply to the new SNMP driver.

The SNMP driver module is used in production by the drivers implementers. If the driver is accepted into the project then this team proposes to support and maintain it in future Ironic development cycles.

The module will be tested and used in production with all available PDU equipment used on-site (APC, Teltronix). The feasibility of implementing a third party CI infrastructure for PDU testing will be investigated and created if possible.

Other collaborators at different sites with PDUs from different vendors would make a valuable contribution to increasing test coverage and qualifying other PDU hardware.

Theoretically, a Tempest suite could be created in which a virtualized PDU was implemented, in the same manner as the fake ssh driver. This approach to test depends on the ability to create an SNMP agent on the test hypervisor and to associate virtual power outlets with VMs. Reviewers thoughts on achieving this concept are welcome.

Documentation Impact

A detailed description of the driver parameters would be needed. A list of tested and qualified PDU hardware would also be helpful. Additionally, any brief notes (in wiki form) on how to configure PDUs from various vendors would be valuable.

References

- PySNMP package on PyPI: https://pypi.python.org/pypi/pysnmp/
- APC PowerNet MIB download (registration may be required): http://www.apc.com/resource/include/techspec_index.cfm?base_sku=SFPMIB403&tab=software
- CyberPower MIB: http://www.cyberpowersystems.com/software/CPSMIB2011.mib
- Eaton Power MIB: http://powerquality.eaton.com/Support/Software-Drivers/Downloads/ePDU/EATON-EPDU-MIB.zip
- Public MIB files repository: http://mibs.snmplabs.com/asn1/

5.15.15 New driver ManagementInterface

https://blueprints.launchpad.net/ironic/+spec/new-management-interface

This blueprint consolidates the work needed for the creation of a new driver interface for management-like operations and a new REST API resource to expose those methods.

Problem description

Almost all Power Interfaces (except the SSH Power Interface) expose a method to set the boot device on their Vendor Passthru interface, this blueprint intends to promote this method to an official interface.

Setting the boot device doesnt fit into the current driver interfaces, even if right now its part of the Vendor Passthru of the Power Interfaces its not actually a power operation, so this blueprint will also include the creation of a new interface called Management Interface where such management operations can be exposed.

As noted in the first paragraph the SSH Power Interface doesnt expose any method to set the boot device, so this is going to be implemented as part of this blueprint. We need all drivers to adhere to the new interface so that its consistent.

As for the API changes, a new management subresource will be added to nodes resource of the REST API and will expose ways to get the supported boot devices and set/get the boot device of a given node. In the future more operations could be added to the management interface, for example: get_sensor_data()[1], update_firmware(), get_firmware_list(), etc

Proposed change

- Create a new ManagementInterface which expose the methods:
  - validate() - To validate driver-specific management information. (E.g if the node has the right credentials already set)
  - set_boot_device() - To set the boot device for a specific node.
  - get_boot_device() - To get the current boot device of a specific node.
  - get_supported_boot_devices() - To get a list of the supported boot devices by that driver.
- Port the set_boot_device() method from the Vendor Passthru interface to the new interface.
- Add a ManagementInterface for ssh, and implement the set_boot_device() for it
- Create a new /nodes/<uuid>/management/boot_device subresource in the REST API to expose ways to the client to {set,get} the boot device for a node.
  - To set the boot device:

    PUT {boot_device: pxe}
    /nodes/<uuid>/management/boot_device[?persistent=<bool>]

    If the request was completed successfully HTTP 204 (No Content) is returned.

    If an invalid or not supported boot device parameter is passed it should return 400 (Bad Request).

    The persistent flag can be set to indicate whether the boot device changes should be applied for the next boot only or to all future boots. By default persistent will be False.
  - To get the current boot device:

    GET /nodes/<uuid>/management/boot_device

    Returns a dictionary with the boot device and if its persistent or not. E.g:

    {boot_device: pxe, persistent: True}

    If boot_device is unknown the value of it will be None (The seamicro python library doesnt seem to expose a way to get the current boot device, only set it[3]).

If persistent is unknown the value of it will be None (The pyghmi method to get the current boot device and doesnt indicate whether its persistent or not[3]. For setting the boot device, its possible to indicate whether its persistent or not).

- To get the supported boot devices:

  GET /nodes/<uuid>/management/boot_device/supported

  This will return a list of all supported boot devices (not the boot order).

- Update the Ironic client and library to support the new API resource.

Alternatives

Continue to use the Vendor Passthru interface.

Data model impact

None

REST API impact

- A new /nodes/<uuid>/management/boot_device subresource will be added to the API, requests to set or get the boot device for a node should go there so that methods like set_boot_device wont be exposed via the vendor_passthru anymore.
- The nodes/<uuid>/validate will include the management interface.

Driver API impact

- The new ManagementInterface will be included in the standardized interfaces group.
- The ipmitool, ipminative, seamicro and ssh drivers are going to be updated to use this new interface.

Nova driver impact

None

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

None

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
lucasagomes

Other contributors:
None

Work Items

- Create the new ManagementInterface base class.
- Port the set_boot_device() method from the Vendor Passthru interface of the ipmitool, ipminative, seamicro drivers to the new interface.
- Implement the missing methods on the ssh, ipmitool, ipminative, seamicro drivers.
- Implement the REST API to expose the new interface.

Dependencies

None

Testing

- Unit tests will be added/updated to cover the changes.
- Tempest tests will be added to Ironic to ensure that the new /nodes/<uuid>/management/boot_device is working properly.

Documentation Impact

The Architecture documentation should be updated to include the new Management Interface.

References

[1] http://specs.openstack.org/openstack/ironic-specs/specs/juno/send-data-to-ceilometer.html [2] https://github.com/seamicro/python-seamicroclient/blob/master/seamicroclient/v2/servers.py#L24 [3] https://github.com/stackforge/pyghmi/blob/master/pyghmi/ipmi/command.py#L123

5.15.16 Send Sensor Data to Ceilometer

https://blueprints.launchpad.net/ironic/+spec/send-data-to-ceilometer

This blueprint will define the sensor data collection interface and implement a driver based on IPMI to collect sensor data and send them to Ceilometer.

Problem description

Ceilometer needs the hardware level performance/status data collection from monitored physical nodes. Ironic owns the IPMI credential, so that is easy for Ironic to get IPMI sensor data and send to Ceilometer via OpenStack common AMQP notification bus.

Proposed change

- Creates a new ironic.driver.base.ManagementInterface common method *get_sensors_data* for gathering hardware sensor data:

  def get_sensors_data(self, task):
  Get the sensors data from physical node

  This method get_sensors_data() will return a dict which is the common data structure to support both IPMI and non-IPMI sensors as below: Sensor Type -> Sensor ID -> Field:Value The sensor current reading value should be found in Sensor Reading field.

  returns
  a consistent format dict of sensor data group by sensor type, which can be processed by Ceilometer.

  Example of actual message: http://paste.openstack.org/show/85130/

- Implements the interface into ipmitool driver to gather data via IPMI command call.

  We call IPMI command ipmitool sdr -v to retrieve sensor data. With -v option we get the entire sensor data including the extended sensor information and the current value in Sensor Reading field.

  Here is an example which is the command return result:

  http://paste.openstack.org/show/63267/

  So we can translate the result to a dict format which is grouped by Sensor Type. For different types, we have different field names. By default, the IPMI node can support the basic three types: Fan, Temperature, and Voltage.

  The non-varying sensors which have no Sensor Reading field will be filtered out and will not be sent to Ceilometer.

- Adds periodic task to conductor which emits notification to Ceilometer by an interval.

  New *periodic_task* method *_send_sensor_data* will be added into *ironic.conductor.manager*. It calls get_sensor_data for all deployed nodes. If get_sensor_data returns None (not implemented) for the node, does not send anything.

  Supports configurations options:

  conductor.send_sensor_data - boolean value, whether to enable collecting and sending sensor data, default value is False.

  conductor.send_sensor_data_interval - Seconds between conductor sending sensor data message to Ceilometer via the AMQP notification bus. The default value is 600.

  Here is the sample message which will be sent to Ceilometer:

  http://paste.openstack.org/show/85053/

- Supports send_sensor_data_types configuration option, where we could list what types of data user want to send, that would allow people to tune which type of data they want to consume and not waste bandwidth sending everything all the time. By default we could send everything. Examples:
  - Default option setting:

    send_sensor_data_types=ALL #by default, send all which has Sensor Reading field, required by Ceilometer.
  - For customization setting:

    send_sensor_data_types=Temperature,Fan,Voltage,Current #which match sensor datas Sensor Type field value.

Alternatives

- An alternative way to do it is that we can enable our ipminative driver to support IPMI sensor data collection, using *pyghmi* lib *get_sensor_data* command method call to access IPMI node.

  However this *get_sensor_data* is one new command introduced recently in latest release still not stable now, I did some testing with physical server, it still does not work, some IPMI sensor raw data can not be parsed by *get_sensor_data* command.

  So we will support ipminative driver later with a separate spec.
- For the *interface* with Ceilometer, we have alternative way to call Ceilometer API directly to avoid less-secure AMQP medium.
- We can allow sensor types to be collected configurable so that it will only collect the sensor data of interest. The default can be all sensors. This can be an optional feature that some Ironic drivers (IPMI driver or non-IPMI vendor drivers) may choose to support later with a separate spec.
- Another way, Ceilometer can pull Ironic API *get_sensors_data* to retrieve the sensor data actively.

Data model impact

None

REST API impact

The management interface will expose the new method *get_sensors_data*. User can call this api to retrieve the sensors data.

Driver API impact

A new function called *get_sensors_data* will be added to the *ironic.driver.base.ManagementInterface*.

Nova driver impact

None

Security impact

We emit notification to Ceilometer via AMQP, AMQP-mediated interactions are assumed to be secure.

Other end user impact

None

Scalability impact

None

Performance Impact

Have some Performance impact about the periodic tasks, they run one after the other in a single green-thread. So periodic tasks like this which poke the BMC will affect the timing of other periodic tasks waiting to run.

Other deployer impact

None

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
Haomeng, Wang(LP ID == whaom)

Other contributors:
Co-work with Ceilometer team developer Chris Dent (LP ID == chdent)

Work Items

- Add a new function interface, *get_sensors_data*, in *ironic.drivers.base.ManagementInterface*.
- Add support in *ipmitool* driver to call IPMI command to get sensor data.
- Add periodic task to conductor to send data by a default interval.

Dependencies

The data notification structure defined by Ironic should be confirmed with Ceilometer team to consume the notifications as well.

So will work with Ceilometer team and fix the structure on demand.

Testing

We will have at least unit tests. But for the IPMI data collection with physical server integration, it is difficult for us to do the *real* testing in CI.

Documentation Impact

Documentation will be extended to explain how it works, and what drivers are supported, and how to enable it.

References

- Ceilometer spec
- Review in progress for sending notification from Ironic.
- Sample data

5.15.17 Support external DHCP providers

https://blueprints.launchpad.net/ironic/+spec/support-external-dhcp

Ironic should allow operators to use an external DHCP server, rather than Neutrons DHCP server.

Problem description

There are a number of reasons a deployer may wish to use an external DHCP server. These include:

- Neutron is not available in the deployers environment.
- The deployer may believe Neutrons DHCP server is not scalable. Neutrons DHCP server uses dnsmasq under the hood, which is known to be unsuitable for large-scale deployments (see links in references section).

Proposed change

Ironic should have a configuration option *dhcp_provider* that specifies which provider to use. All DHCP providers will implement a DHCP provider interface, which exposes a method to update a ports DHCP attributes.

Drivers will no longer call the DHCP provider directly, but instead request a DHCP provider from a factory, which will return the provider specified in the configuration options.

The two initial provider implementations will be neutron and none.

Neutron functionality that does not relate to DHCP (ie. updating a ports MAC address) will not be moved.

Alternatives

It would be simpler in the short term to have a boolean *use_neutron_for_dhcp* config that would tell Ironic to either use Neutron or do nothing. This would duplicate the initial functionality of the proposed change. However, new DHCP providers would be difficult to add, and there would also be more work for the deploy driver.

Data model impact

None.

REST API impact

None.

Driver API impact

None.

Nova driver impact

None.

Security impact

None.

Other end user impact

None.

Scalability impact

This will only make Ironic more scalable, as it will expose a configuration in which Ironic no longer interacts with Neutron.

Performance Impact

None.

Other deployer impact

A config option *dhcp_provider* will be added that starts with two options, neutron and none.

There will be no immediate impact on deployers, as *dhcp_provider* will default to neutron, which does not change behavior.

However, deployers that wish to use an external DHCP provider must set the config option and also deploy a DHCP service.

Developer impact

None.

Implementation

Assignee(s)

Primary assignee:
jroll

Other contributors:
JoshNang ellenh

Work Items

- Add *dhcp_provider* config option.
- Move Neutron API behind a DHCP provider interface.
- Implement none DHCP provider.
- Implement a DHCP provider factory.
- Add Tempest tests

Dependencies

None.

Testing

Tempest tests should be written that use an external DHCP provider. dnsmasq is already installed in devstack, so this could be accomplished by running dnsmasq standalone, rather than via Neutron.

Documentation Impact

Documentation will need to be written for this config option. This should include:

- What the config option does.
- How to configure an external DHCP server to work with Ironic.
- A diagram of what the control plane looks like for both dhcp_provider options.

References

- Dnsmasq provides network infrastructure for small networks: http://www.thekelleys.org.uk/dnsmasq/doc.html

5.15.18 Swift Temporary URLs

https://blueprints.launchpad.net/ironic/+spec/swift-temp-urls

Ironic needs an option to download images securely from Swift without passing the admin auth token all over the place. Using Swift temporary URLs, the conductor could create an expiring URL that has full access to download only a specific image. This is very helpful for the Ironic Python Agent, because we dont want to send admin auth tokens to all the ramdisk agents, opening a potential security vulnerability, but they still need a way to download images.

Problem description

Today, Ironic downloads images to write out directly from Glance. In the current PXE driver, there isnt a security concern because the images are downloaded to the conductor and tokens do not have to be passed around. With the Ironic Python Agent and other drivers like iLO, images need to be downloaded into the ramdisk. To download from Glance, an auth token would need to be sent with the request. In Ironics case, that would be an admin token, which would allow the agent to have admin control of other services. Many other services/drivers in Ironic may want the ability to download from Swift but limit what a service has access to in a similar fashion.

There is also a scaling benefit to download directly from Swift, rather than using Glance as an intermediary. The Swift cluster can be scaled to deal with larger loads without much additional load on the Glance cluster.

Proposed change

The ability to create a signed temporary URL for an image registered in Glance and stored in Swift which can be downloaded without additional credentials.

- Given the image info returned by Glances show command and a previously set temporary URL key in Swift, a temporary URL will be created using the Swift client library and returned. It will either use the image infos direct_url property if set, or a set of config options in Ironic to generate the URL.
- The key is generated using a hash of the HTTP methods, the timeout, the image path, and the shared key, which is appended to the end of the Swift object URL, along with the timeout. No call to Swift or Glance is necessary, because it is signing the URL with a shared secret key. The actual create of the temp url is handled by python-swiftclient. Reference: [1]
- The method will support either constructing the URL from direct_url in the Glance image info or a set of config options set in Ironic.
- If direct_url is not enabled, the deployer will need to set config option for the scheme to talk to Swift with (http, https), the Swift endpoint URL, the path (which includes the API version and the tenant ID), and the container that the Glance images are stored in.

Alternatives

- Enable passing admin auth tokens to agents or to host the images in a way that the agent would be able to download them.
- A deployer could pass Swift URLs with the username and password in the URL itself, which would be less insecure than admin auth tokens, but still insecure.
- The images could also be hosted by any web server, and that URL could be passed to the agent.
- Ideally, this code should live in keystone-client or oslo, so other projects could benefit from temporary URLs as well. We are proposing it in Ironic for now, because we need to use it sooner rather than later, but it should be moved later.

Data model impact

None

REST API impact

None

Driver API impact

None

Nova driver impact

None

Security impact

- This code requires a Swift temporary URL key to be set. If this key is leaked, it could allow a bad actor to allow public access to anything they have access to in Swift.
- A temporary URL gives complete public access to possibly private images until the expiration.
- The only allowed HTTP methods are GET and HEAD, so bad actors will not be able to modify the images themselves.

Other end user impact

None

Scalability impact

This change should allow better scaling. Swift is highly scalable and designed to allow you to download large images. This will lessen the load on the Glance cluster. The main bottleneck would be the Swift - nodes network links. They could easily be saturated if a large number of nodes attempt to download an image at the same time.

Performance Impact

The code is relatively small and self contained. It requires the result of a Glance image-show command to make the temporary URL, however that Glance call may already be required (in the agent driver, for example). The Glance image-show call in Ironic is here: https://github.com/openstack/ironic/blob/master/ironic/common/glance_service/base_image_service.py#L176. The temporary URL works via signing the direct-URL (from Glance image-show) or a URL constructed with the proposed config options with a shared private key, so apart from this one Glance command, the temporary URL can be generated. There are no additional database calls required.

Other deployer impact

- Deployers will need to set up a Swift cluster and configure Glance to use it if they want to take advantage of temporary URLs.
- Either direct_url needs to be enabled in the Glance cluster or a list of config options needs to be set to translate Glance image IDs to Swift URLs. To configure direct_url: [3]

Required config options:

- swift_temp_url_key: This is the shared private key. It needs to be set up on the Swift cluster prior to use. To set up a temporary URL key: [1]
- swift_temp_url_duration: How long the Swift temporary URL is good for. Should be set relatively low to prevent allowing images to become public. 5 minutes should be enough time to start the download and minimize security concerns about the URL getting out. The duration will be specified in seconds.

Required if direct_url is not enabled in Glance, and the options have no defaults:

- swift_endpoint_url: The scheme, hostname, and optional port of the Swift cluster. For example, http://example.com:8080.
- swift_path: The API version and tenant ID of the cluster and container that Glance images are stored in. For example, /v1/TENANT_USER_TENANT_UUID.
- swift_backend_container: The Swift container in which Glance stores its images.

Developer impact

This change will allow other driver developers to use Temporary URLs or at least provide them as an option.

Implementation

Assignee(s)

Primary assignee:
JoshNang

Work Items

- Address comments on the current patch.
- Add tempest tests

Dependencies

- Merging of the python-swiftclient tempurl patch [2]
- Adding python-swiftclient to Ironics requirements.txt

Testing

- Initially, only unit-testing will be conducted
- This change will also be tested as part of the Ironic Python Agent integration testing. It is challenging to test on its own, as it does not present any external APIs for Tempest to test.

Documentation Impact

To implement this feature, Glance using Swift as the image datastore is required. Swift will also require the tempurl middleware to be configured for the temporary URLs to work. Swift will need a tempurl key to be set before this feature can be used, and it needs to be set in Ironic as swift_temp_url_key. Glance either needs to be configured to provide direct_url or the operator must set the following config options:

- swift_endpoint_url
- swift_path
- swift_backend_container

The actual signing code is proposed in python-swiftclient.

Detailing how to use Temporary URLs with a Swift cluster and how to configure Glance to use direct_url would be helpful for deployers. Drivers that utilize the change should link to these requirements.

References

- Swift Temporary URLs: http://docs.openstack.org/trunk/config-reference/content/object-storage-tempurl.html
- Proposed patch in python-swiftclient: https://review.openstack.org/#/c/102632/
- Glance direct_url configuration: https://github.com/openstack/glance/blob/master/etc/glance-api.conf#L89

5.15.19 UEFI support for Ironic deploy drivers

https://blueprints.launchpad.net/ironic/+spec/uefi-boot-for-ironic

This spec proposes to add support for UEFI based deployments on baremetal nodes.

Problem description

Most of the new hardware comes with UEFI boot mode, which has several technical advantages over the traditional BIOS system. The new servers also have a compatibility support module(CSM), which provides BIOS compatibility.

Currently there is no provision in Ironic to let a user select and deploy a baremetal node having capability to boot in UEFI boot mode.

Proposed change

Preparing the environment

1. The operator sets the baremetal node to boot in the desired mode - bios or uefi.
2. The operator can inform the boot mode to ironic using the `capabilities` property of the node. The operator may add a new capability `boot_mode=uefi` or `boot_mode=bios` in `capabilities` within `properties` of the node.

Preparing flavor for boot mode selection

The `extra_specs` field in the nova flavor may be used for selection of a machine with the desired boot mode. The operator may create a flavor with `boot_mode=bios` or `boot_mode=uefi` to select a baremetal node set to exactly same `boot_mode`.

If `boot_mode` is not present in `extra_specs` of nova flavor, then nova scheduler may give a baremetal node configured in any boot mode.

Pxe deploy driver changes

Changes required in PXE deploy driver to perform UEFI boot mode deploy:

- Add new pxe config options:
  - `uefi_pxe_bootfile_name`: specify the efi bootloader to be used.
  - `uefi_pxe_config_template`: specify the respective efi bootloader config template.
- Prepare pxe config for UEFI, by reading `uefi_pxe_bootfile_name` and its corresponding `uefi_pxe_config_template`.
  - As of now I will use elilo.efi boot loader.

  - elilo.efi bootloader requires the configuration to be named after the ip-address assigned by the DHCP server. It does not recognize mac-address named config files.

- Update neutron port DHCP extra opts with correct boot file for UEFI boot:
  - `bootfile-name`: value should be fetched from `uefi_pxe_bootfile_name`

Other deploy driver changes

Other deploy drivers may handle the uefi boot option in their deploy driver code to support UEFI boot mode.

Some deploy drivers (like iLO driver which uses proliantutils library) will be able to change the boot mode itself, rather than relying on admin to change the boot mode. Such deploy drivers may add functionality to change the boot mode dynamically on a request. Admin will just need to document the `boot_mode` that the machine is supposed to be used (Admin need not do #1 in the section Preparing the environment above).

Overall flow

1. Ironic virt driver picks up `boot_mode` (if available) from the `capabilities` field of the Ironic node and registers it as a capability of the hypervisor. (*NOTE*: The functionality to do this is already available in the proposed nova-ironic virt driver).
2. User may select a flavor having `boot_mode` specified in its `extra_specs`.
3. `ComputeCapabilities` filter of nova scheduler matches the `boot_mode` (if available) against the `boot_mode` of the node.
4. The deploy driver reads the `boot_mode` from the `capabilities` property of the node and then makes appropriate changes to deploy process to deploy and boot the baremetal node in the required `boot_mode`.

Alternatives

- Pxe driver can support different efiboot loaders: syslinux.efi, grub, etc.
  - Different bootloaders will have different ways of preparing their configuration files.
  - Though pxelinux and syslinx.efi have same configuration changes, but syslinux.efi is not yet available on ubuntu 12/13/14. syslinux.efi is scheduled for ubuntu utopic release.
  - We will support only elilo.efi at the moment. We can later add support to other efi bootloader.
- There could be other vendor specific boot modes, or other boot options with-in BIOS/UEFI boot modes. We can support them incrementally on top of these standard boot modes.
- UEFI with local HDD boot:
  - As of now there is no support for local HDD boot with pxe driver.
  - When the deploy driver add support for local HDD boot with BIOS mode, they have to consider adding support for UEFI as well, if the driver support UEFI boot mode.
- Selecting a partition vs whole-disk image for deploy:
  - A partition image can be installed using both bios and UEFI boot mode, where as a whole-disk image may ask for a specific boot_mode for deploy.
  - With a partition image we need not specify the required boot_mode.

  - As of now pxe driver supports only partition images, when we add support for deploy with whole-disk image, we need to specify the required boot mode for that image and pass it on to deploy driver.

- Using IronicBootModeFilter to schedule both uefi and bios boot mode requests:
  - with ComputeCapabilities filter we can schedule predefined boot_mode on a node, which is capable of both uefi and bios boot modes. For example, if we set boot_mode:uefi in capabilities node property, on a node which is capable of both uefi and bios boot modes, then scheduler will not pick this node if user has specified bios in nova flavor.
  - With IronicBootMode filter, we can schedule both uefi and bios boot mode request on the same node which is capable of both boot_modes.

Data model impact

None.

REST API impact

None.

Driver API impact

None.

Nova driver impact

None.

Security impact

This feature will enable a later enhancement to support uefi secure boot.

Other end user impact

- User can trigger a UEFI boot mode deploy by selecting a flavor with `boot_mode` in the `extra_specs` field.
- Ironic nodes should have additional properties to support UEFI based deploy.

Scalability impact

None.

Performance Impact

None.

Other deployer impact

- Operator may to set the boot mode of baremetal node to the desired one manually.

- Operator may set a new capability `boot_mode` in `capabilities` within `properties` of the ironic node. For example, the user may add `capabilities:boot_mode=uefi` for a baremetal node which is configured for uefi boot mode.
- Copy UEFI bootloader (elilo.efi) under tftp root directory.
- Set pxe configuration parameters: `uefi_pxe_bootfile_name` and `uefi_pxe_config_template`

Developer impact

Other deploy drivers may handle the uefi boot option in their deploy driver code to support UEFI boot mode.

Implementation

Assignee(s)

Faizan Barmawer.

Work Items

1. Implement the code changes for supporting uefi boot mode in pxe driver.
2. Other drivers can implement changes required to support UEFI mechanism.

Dependencies

None.

Testing

Unit tests will be added for the code.

Documentation Impact

Documentation should be modified to instruct admin to place efi bootloader in tftp root and ironic node property updation.

References

http://sourceforge.net/projects/elilo/ http://webapp5.rrz.uni-hamburg.de/SuSe-Dokumentation/packages/elilo/netbooting.txt

5.15.20 Bare Metal Trust Using Intel TXT

This spec was proposed in the Liberty cycle.

See *Bare Metal Trust Using Intel TXT*.

5.15.21 Cisco UCS PXE driver

This spec was proposed in the Liberty cycle.

See *Cisco UCS PXE driver*.

5.15.22 Add enroll state to the state machine

This spec was proposed in the Liberty cycle.

See *Add enroll state to the state machine*.

5.15.23 Move Ironic to a feature-based release model

This spec was proposed in the Liberty cycle.

See *Move Ironic to a feature-based release model*.

5.15.24 Switch periodic tasks to the Futurist library

This spec was proposed in the Liberty cycle.

See *Switch periodic tasks to the Futurist library*.

5.15.25 In-band RAID configuration using agent ramdisk

This spec was proposed in the Liberty cycle.

See *In-band RAID configuration using agent ramdisk*.

5.15.26 iPXE Dynamic Configuration

This spec was proposed in the Liberty cycle.

See ../approved/ipxe-dynamic-config.

5.15.27 iPXE to use Swift Temporary URLs

This spec was proposed in the Liberty cycle.

See *iPXE to use Swift Temporary URLs*.

5.15.28 iRMC Virtual Media Deploy Driver for Ironic

This spec was proposed in the Liberty cycle.

See *iRMC Virtual Media Deploy Driver for Ironic*.

5.15.29 New driver interface for RAID configuration

This spec was proposed in the Liberty cycle.

See *New driver interface for RAID configuration*.

5.15.30 Open CloudServer (OCS) power driver

This spec was proposed in the Liberty cycle.

See *Open CloudServer (OCS) power driver*.

5.15.31 Pluggable network providers

This spec was proposed in the Liberty cycle.

See *Pluggable network providers*.

5.15.32 Boot Interface in Ironic

This spec was proposed in the Liberty cycle.

See *Boot Interface in Ironic*.

5.15.33 Nodes tagging

This spec was proposed in the Liberty cycle.

See *Nodes tagging*.

5.15.34 Override PXE options via Glance property

This spec was proposed in the Liberty cycle.

See *Override PXE options via Glance property*.

5.15.35 Petitboot boot driver

This spec was proposed in the Liberty cycle. In the Rocky cycle it was moved to the backlog folder as it is not presently being worked upon with-in the community.

See ../backlog/petitboot-boot-driver.

5.15.36 Make ilo drivers standalone in ironic by removing swift dependency

This spec was proposed in the Liberty cycle.

See *Make ilo drivers standalone in ironic by removing swift dependency*.

5.15.37 Determinable supported boot device list

This spec was proposed in the Liberty cycle.

See *Determinable supported boot device list*.

5.15.38 UEFI Secure Boot support for pxe_iLO driver

This spec was proposed in the Liberty cycle.

See *UEFI Secure Boot support for pxe_iLO driver*.

5.15.39 Client Caching Of Negotiated Version

This spec was proposed in the Liberty cycle.

See *Client Caching Of Negotiated Version*.

5.15.40 Wake-On-Lan (WOL) power driver

This spec was proposed in the Liberty cycle.

See *Wake-On-Lan (WOL) power driver*.

5.15.41 Bare Metal Trust Using Intel TXT

https://bugs.launchpad.net/ironic/+bug/1526280

This uses Intel TXT[4], which builds a chain of trust rooted in special purpose hardware called Trusted Platform Module (TPM)[3] and measures the BIOS, boot loader, Option ROM and the Kernel/Ramdisk, to determine whether a bare metal node deployed by Ironic may be trusted.

Problem description

The bare metal tenant has the ability to introduce rootkits and other malware on the host. Prior to releasing the host to a new tenant, it is prudent to ensure the machine is in a known good state.

Using Intel TXT[4], the TPM[3], Trusted Boot[1], and remote authentication[2], it is possible to confirm that the BIOS, boot loader, Option ROM, and the Kernel/Ramdisk are all in a known good state.

Proposed change

Add a new boot mode, trusted boot:

- Read value capabilities:trusted_boot from flavor. Pass boolean value trusted_boot to ironic.drivers.modules.deploy_utils.switch_pxe_config(). Switch to trusted_boot section.
- Add a new section trusted_boot in PXE Configuration. It will make use of mboot.c32 which supports multiple loading. It loads TBOOT first. TBOOT will measure Kernel/Ramdisk before loading them. PXE config template:

```
label trusted_boot
kernel mboot
append tboot.gz --- {{pxe_options.aki_path}} root={{ ROOT }} ro text
{{ pxe_options.pxe_append_params|default("", true) }} intel_iommu=on
--- {{pxe_options.ari_path}}
```

Alternatives

Secure Boot[5] is used for the same purpose. The main difference is secure boot will verify the signature before executing while trusted boot uses a hardware root of trust and can be configured to verify each component before executing or execute all components and capture measurements (aka extended hash computations) for post verification. So if a node is changed, trusted boot will still boot it up but give a warning to users. Secure boot will not boot it up at all.

They are complementary, both making the cloud more secure. It is recommended to boot nodes with secure boot under uefi and boot nodes with trusted boot under legacy BIOS. The next step is to combine them together but that is out of the scope of this spec.

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

RPC API impact

None

Driver API impact

None

Nova driver impact

Will pass the extra_spec capabilities:trusted_boot=True to Ironic

Ramdisk impact

N/A

Security impact

Increased confidence in bare metal nodes being free of rootkits and other malware. Intel TXT and TPM are leveraged.

Other end user impact

None

Scalability impact

Our experiments indicate handling concurrent attestation requests is linear in the number of requests. Attestation occurs on the node release path, and thus is not latency sensitive.

Performance Impact

There is an extra attestation step during trusted boot which spends several seconds. But for bare metal trust no dynamic attestation requests are entertained. So this is a non-issue.

Other deployer impact

- Create a special flavor with capabilities:trusted_boot=True
- Set `trusted_boot:True` as capability in node.properties.
- **Additionally two items need to be provided with tftpboot/httpboot folder**
  - mboot.c32 - Support multiple loading from /usr/lib/syslinux/mboot.c32
  - tboot.gz - a pre-kernel module to do measurement.

- Set up each machine, enable Intel TXT, VT-x and VT-d and take ownership of the TPM, reboots, and captures the platform configuration register (PCR) values. This is to create the whitelist values that will be registered in the attestation service at initialization time.
- Set up an OAT-Server and create the whitelist with all known types of hardwares from previous step.
- Create customized images with OAT-Client.
- Run a customized script to verify the trust state of nodes when creating instances.

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
tan-lin-good

Work Items

- Add trusted_boot section to pxe_config.template
- Support trusted_boot flag and switch to trusted_boot.
- A dib element to create customized images.

Dependencies

- TBOOT[1]
- OAT[2]
- Hardware Support: TPM and Intel TXT

Testing

Will add unit tests. Planning on adding third party hardware CI testing.

Upgrades and Backwards Compatibility

None. Backwards compatibility is achieved by not requesting trusted bare metal. Custom tenant images are accommodated by deploying an initial standard image that has the OAT client embedded. Today Fedora releases come bundled with the OAT client. This solution approach, while increasing the number of boots preserves us from having to doctor the tenant image by way of injecting the OAT client into the same, or requiring that bare metal users provide images with an OAT client included.

Documentation Impact

Will document usage and benefits. Here is a doc for the technical detail of Bare metal trust: https://wiki.openstack.org/wiki/Bare-metal-trust

References

1. http://sourceforge.net/projects/tboot/
2. https://github.com/OpenAttestation/OpenAttestation
3. http://en.wikipedia.org/wiki/Trusted_Platform_Module
4. http://en.wikipedia.org/wiki/Trusted_Execution_Technology
5. https://review.openstack.org/#/c/135228/
6. http://docs.openstack.org/admin-guide-cloud/compute-security.html#trusted-compute-pools

5.15.42 Cisco IMC PXE Driver

https://blueprints.launchpad.net/ironic/+spec/cisco-imc-pxe-driver

This spec proposes adding a new driver which provides out-of-band non-ipmi based control of UCS C-Series servers through CIMC.

Problem description

Current drivers only allow for control of UCS servers via either IPMI or UCSM, the Cisco UCS C-Series operating in standalone mode can also be controlled via CIMC using its http/s XML API. This provides finer control over the server than IPMI can, and doesnt require the extra infrastructure that UCSM needs.

Proposed change

Power and Management interfaces will be created that understand how to talk to CIMC, and these will be used in conjunction with the PXE boot interface, ISCSI deploy interface and Agent deploy interface to create pxe_cimc and agent_cimc drivers.

The CIMC Power interface will inherit the base PowerInterface and implement:

- get_power_state
- set_power_state
- reboot
- get_properties
- validate

The CIMC Management interface will inherit the base ManagementInterface and implement:

- get_properties
- validate
- get_supported_boot_devices
- get_boot_device
- set_boot_device
- get_sensors_data - This will raise NotImplemented

Alternatives

The alternatives are to use the pxe_ipmi or agent_ipmi driver to control the UCS C-Series via IPMI, or install the infrastructure to manage these servers with UCSM and use the pxe_ucs or agent_ucs driver.

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

When enrolling a node into ironic a user must provide:

- cimc_address - CIMC IP Address
- cimc_username - CIMC Username
- cimc_password - CIMC Password

Additional properties added to ironic.conf in a [cimc] section are:

- max_retry: maximum times to retry any power operation, default: 6
- action_interval: the time to wait in-between power operation retries

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
sambetts

Other contributors:
None

Work Items

- Write CIMC Power Interface and respective unit tests
- Write CIMC Management Interface and respective unit tests
- Create drivers from new and existing interfaces
- Create configuration documentation for pxe_cimc and agent_cimc

Dependencies

This driver requires this installation of the ImcSDK on the node where the ironic conductor will be running.

Testing

Unit tests for the Power and Management interfaces will be provided. Functional testing will be added in the future.

Upgrades and Backwards Compatibility

There should be no compatibility issues introduced by this change.

Documentation Impact

- Writing configuration documentation.
- Updating Ironic documentation section Enabling Drivers: http://docs.openstack.org/developer/ironic/deploy/drivers.html with pxe_cimc and agent_cimc driver related instructions.

References

Cisco Imc Python SDK v0.7.1: https://communities.cisco.com/docs/DOC-37174

5.15.43 Cisco UCS PXE driver

https://blueprints.launchpad.net/ironic/+spec/cisco-ucs-pxe-driver

This blueprint proposes adding new driver that supports deployment of Cisco UCS Manager (UCSM) managed B/C/M-series servers.

In this blueprint servers, nodes are used interchangeably that denotes Cisco UCS B/C/M-series servers.

Problem description

Current Ironic drivers require IPMI protocol to be enabled on all Cisco UCS B/C/M-series servers in order to manage power operations. For security reasons from UCS Manager version 2.2.2 IPMI protocol is disabled by default on all servers.

Instead of using IPMI protocol, this blueprint proposes new driver to manage Cisco UCS B/C/M-series servers using Cisco UCS PySDK.

Proposed change

New power and management interfaces will be added as part of pxe_ucs driver. This driver uses the Cisco UCS PySDK for communicating with UCSM.

This driver uses

- pxe.PXEDeploy for PXE deployment operations
- ucs.power.Power for power operations
- ucs.management.UcsManagement for management interface operations

UCS Manager provides python SDK with which user can perform various operations, like controlling the power of nodes, enabling the ports, associating the servers etc. Physical and Logical entities in UCSM are represented as ManagedObject.

- Power management:

  Controlling the power is similar to modifying the property of the corresponding ManagedObject. LsPower is the ManagedObject that represents the Power of service-profile. As part of managing the power, this provider modifies state property of LsPower ManagedObject. UCSM takes care of the rest.

- Management Interface:

  This interface allows the user to get and set the boot-order on UCS B/C/M servers. LsbootDef is the ManagedObject that represents the boot-order of service-profile. This interface reads and updates LsbootDef ManagedObject appropriately for get and set boot-device operations. get_sensor_data() implementation is not in scope of this spec. A separate spec will be submitted.

Alternatives

The IPMI Pxe driver could be used with Cisco UCS B/C/M-series servers, if IPMI protocol is explicitly enabled overriding the default settings in UCS Manager.

Data model impact

None

RPC API impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

Driver API impact

None

Nova driver impact

None

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

The following driver_info fields are required while enrolling node into ironic:

- ucs_address: UCS Manager hostname/ip-address
- ucs_username: User account with admin/server-profile access privilege
- ucs_password: User account password
- ucs_service_profile: service_profile DN (DistinguishedName) being used for this node.

The following parameters are added in to the newly created [ucs] section in the ironic configuration file which is typically located at /etc/ironic/ironic.conf.

- max_retry: maximum number of retries, default value is set to 5.
- action_timeout: seconds to wait for power action to be completed default value is 30 seconds, there is no explicit maximum limit.

Developer impact

None

Implementation

Assignee(s)

Primary assignee: saripurigopi

Other contributors: vinbs

Work Items

- Add new pxe_ucs driver, extending power and management interface APIs.
- Writing and unit-test cases for pxe_ucs driver.
- Writing configuration documents.

Dependencies

This driver requires Cisco UCS Python SDK installed on the conductor node.

Testing

Unit-tests will be implemented for new pxe_ucs driver. tempest test suite will be updated to cover the pxe_ucs driver. Continuous integration (CI) support will be added for Cisco UCS B/C/M series servers.

Upgrades and Backwards Compatibility

This driver will not break any compatibility with either on REST API or RPC APIs.

Documentation Impact

- Writing configuration documents.
- Updating Ironic documentation section Enabling Drivers: http://docs.openstack.org/developer/ironic/deploy/drivers.html with pxe_ucs driver related instructions.

References

Cisco UCS PySdk:https://github.com/CiscoUcs/UcsPythonSDK

5.15.44 Deprecate the bash ramdisk

https://blueprints.launchpad.net/ironic/+spec/deprecate-bash-ramdisk

This spec is a continuation of the blueprint ipa-as-default-ramdisk implemented in the Kilo release. This spec intends to deprecate deployments using the bash script ramdisk.

Problem description

The bash ramdisk is still supported by the drivers prefixed with `pxe_` without any deprecation message. In the Kilo release it was agreed that we should stop supporting the bash ramdisk in the future and we worked on making the IPA ramdisk supported by all drivers in tree.

Also, the bash ramdisk is already lagging behind support for some features, for example cleaning only works with IPA. So now we should start dropping the support for that ramdisk.

Proposed change

We can not simply delete the code that the bash ramdisk uses, therefore we should start adding deprecation messages on the `deploy-ironic` element from diskimage-builder and in the vendor passthru methods `pass_deploy_info` and `pass_bootloader_install_info` which are used by the bash ramdisk to pass the deployment information to Ironic.

Apart from the deprecation messages this spec also proposes freezing the features for the bash ramdisk. No new features should be added to it (like we did to include support for local boot), only bug fixes will be accepted.

Devstack and tempest jobs should also be updated to not use the bash ramdisk anymore.

The element in diskimage-builder and the deprecated code in Ironic should be removed in the Mitaka release cycle of OpenStack.

Alternatives

Continue to support the bash ramdisk for a longer time.

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

Deployer should start replacing the bash ramdisk with the IPA ramdisk. Theres no new configuration needed for it, its a drop-in replacement.

Developer impact

Developers wont be allowed to include any new features to the bash ramdisk, only bug fixes.

Implementation

Assignee(s)

Primary assignee:
lucasagomes <lucasagomes@gmail.com>

Other contributors:
everyone

Work Items

- Update Devstack and tempest to use the IPA ramdisk instead of the bash ramdisk.
- Add deprecation messages on the diskimage-builder `deploy-ironic` element and vendor passthrus `pass_deploy_info` and `pass_bootloader_install_info`.
- Stop accepting new features for the bash ramdisk (code reviews and spec review).
- In the Mitaka release cycle remove the element from *diskimage-builder* and the code that supports the bash ramdisk in Ironic.

Dependencies

None

Testing

Unittests will be added.

Upgrades and Backwards Compatibility

None

Documentation Impact

The documentation should be updated to say that the bash ramdisk is deprecated and the examples should now use IPA instead.

References

5.15.45 DRAC vendor passthru for BIOS settings management

https://blueprints.launchpad.net/ironic/+spec/drac-bios-mgmt

This spec will expose the vendor passthru methods needed to let external services get, set, and commit changes to the BIOS settings. This spec will assume that the external service knows exactly what it is doing with the specific hardware it will manage, and it will not attempt to standardize, normalize, or simplify the exposed settings in any way beyond some minimal convenience measures and what is needed to map between XML and JSON.

Problem description

Tuning a server for a particular workload and power consumption profile involves many different parameters to be tweaked, several of which must be tuned at the level of the system firmware. On Dell systems, doing that out-of-band involves talking to the DRAC to have it make changes on your behalf.

Proposed change

To expose these changes, I propose to add 4 vendor passthrough methods to the drac driver:

- `get_bios_config` - Gather the current BIOS configuration of the system from the drac and return it. This will return a JSON structure that contains the current BIOS configuration of the system. It has no parameters.
- `set_bios_config` - Queue up changes to the BIOS configuration for later committal. This will accept the same JSON data structure that `get_bios_config` returns. It will raise an exception if

any of the changed settings does not validate as a proper change, if you try to change a readonly setting, or if the changes cannot be queued up in the DRAC for some other reason.

- `commit_bios_config` - Commit a set of queued up BIOS configuration changes, and schedules a system reboot if is needed to commit the changes.
- `abandon_bios_config` - Abandon a set of queued up BIOS configuration changes.

For this spec, only changes that can be made though the DCIM Bios and Boot Management Profile will be considered, although there are many other BIOS and hardware parameters that can be changed by the DRAC.

Alternatives

Create a common BIOS interface which most vendors will agree to. This is a longer term solution which should replace or drive this implementation in the future.

Data model impact

None.

REST API impact

Four new API calls:

- `get_bios_config` - a GET method. Takes no args, returns a blob of JSON that encapsulates the BIOS configuration parameters of the system.
- `set_bios_config` - a POST method. Takes a blob of JSON that has the same format and settings returned by get_bios_config. Returns the following status codes:
  - 200 if the set of proposed new parameters passed validation and was accepted for further processing.
  - 204 if the set of proposed new parameters passed validation, but did not actually change the BIOS configuration.
  - 409 if the set of proposed new parameters contains a parameter that cannot be set to the requested value, either because the parameter is read-only or the proposed new parameter is not valid.
  - 403 if there are already proposed changes in the process of being committed.
- `commit_bios_config` - a POST method. Takes no args, commits the changes made by `set_bios_config` calls and schedules a system reboot to apply the changes if needed. Returns the following status codes:
  - 202 if a commit job was created, and the system requires a reboot.
  - 200 if the settings were committed without needing a reboot.
  - 204 if there were no settings that needed to be committed.
  - 403 if there are already proposed changes in the process of being committed.
- `abandon_bios_config` - a POST method. Takes no arguments, and abandons any changes made by `set_bios_config` calls that have not been committed by a `commit_bios_config` job. Returns the following status codes
  - 200 if the proposed changes were successfully dequeued.

- 204 if there were no proposed changes to dequeue
- 403 if the proposed changes are already in the process of being committed.

RPC API impact

None

Driver API impact

None

Nova driver impact

None for now. If this spec gets generalized to other drivers, we will need to figure out what (if any) capabilities should be exposed to Nova.

Security impact

It is quite possible to render a system unbootable through this API, as it allows you to enable and disable a wide variety of hardware and mess with the boot order indiscriminately.

Other end user impact

None

Scalability impact

None

Performance Impact

None.

Other deployer impact

None

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
victor-lowther

Work Items

- Create and implement DracVendorPassthruBios class

Dependencies

- This feature depends on the python bindings of the OpenWSMAN library which we already use for the rest of the DRAC driver.
- This feature requires 11th or higher generations of Dell PowerEdge servers.

Testing

- Unit tests
- 3rd-party CI: I will try to implement it in parallel with implementing this driver, provided I can source sufficient internal resources and appropriate network connectivity.

Upgrades and Backwards Compatibility

None expected, and there should be no stability guarantee for this API.

Documentation Impact

User documentation should mention this vendor passthrough API.

References

DCIM Bios and Boot Management Profile

5.15.46 Add enroll state to the state machine

https://blueprints.launchpad.net/ironic/+spec/enroll-node-state

This blueprint introduces a new state called `enroll`, which we previously agreed to introduce in the new state machine spec.

Problem description

Currently nodes on creation are put into `available` state, which is designed as a replacement for `NOSTATE`. Such nodes will instantly be available to Nova for deployment, provided they have all the required properties.

However, the new state machine lets the operator perform inspection on a node and zapping of a node. The state machine allows for them to be done before a node reaches the `available` state.

Even worse, the cleaning feature introduced in Kilo cycle should also happen before a node becomes `available`.

Proposed change

- Add a new state `enroll`, from which a node can transition into the `manageable` state by an action called `manage`.
- `manage` transition will cause power and management interfaces to be validated on the node. Also an attempt will be made to get the power state on a node to actually verify the supplied power credentials.

On success, the node will go to the `manageable` state. On failure, it will go back to the `enroll` state and `last_error` will be set.

- Disable the sync_power_state for nodes in the `enroll` state, as nodes in this state are not expected to have valid power credentials.
- Introduce a new API microversion, making newly created nodes appear in the `enroll` state instead of the `available` state.

  After that the client-server interaction will be the following:

  - new client (with new API version) + new server: nodes appear in the `enroll` state.
  - new client + old server: client gets a response from the server stating that node is in `available` (or `none`) state. Client issues a warning to the user.
  - old client (or new client with old API version) + new server: due to versioning the node will appear in `available` (or `none`) state.
- Document that we are going to make a breaking move to the `enroll` state by default.
- Update DevStack gate to explicitly request the new microversion and fix the tests.
- Release a new version of `ironicclient` defaulting to this new microversion. Clearly document this breaking change in upgrade notes.

Alternatives

We can ask people to manually move nodes to the `manageable` state before inspection or zapping. We also wont validate power and management interfaces.

Data model impact

None

State Machine Impact

- `enroll` becomes a valid node state with transitions to:
  - `verifying` via `manage` action
- `verifying` becomes a valid transient node state with transitions to:
  - `manageable` on `done`
  - `enroll` on `fail` and `last_error` will be set.

REST API impact

- Add new API microversion. When it is declared by client, node creation API should default to creating nodes in the `enroll` state.

Client (CLI) impact

- New release of client will be issued defaulting to the new microversion (and breaking many flows).

RPC API impact

None

Driver API impact

None

Nova driver impact

- Double check that Nova driver wont use nodes in `enroll` state
- Sync `nova/virt/ironic/ironic_states.py` for the sake of consistency

No functionality impact expected.

Security impact

None expected

Other end user impact

With the new microversion, nodes will appear in the `enroll` state. Two more steps should be taken to make them available for deploy: `manage` and `provide`. Cleaning will happen, if enabled.

Scalability impact

None

Performance Impact

None

Other deployer impact

See *Upgrades and Backwards Compatibility*.

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
Dmitry Tantsur, IRC: dtantsur, LP: divius

Other contributors:
None

Work Items

- Create new states and transitions
- Introduce new microversion with node defaulting to `enroll` on creation
- Make sure our tests do not break (fix devstack etc)
- Default ironicclient to the new microversion

Dependencies

None

Testing

- Tempest tests should be modified to test `enroll` state.

Upgrades and Backwards Compatibility

- Change is backwards compatible, while its not the default in ironicclient.
- Once new microversion is the default in ironicclient, it will break existing flows, when explicit microversion is not in use.

Documentation Impact

- Working with the new state and the transition should be documented
- Upgrade notes should be updated

References

5.15.47 Move Ironic to a feature-based release model

https://blueprints.launchpad.net/ironic/+spec/feature-based-releases

At the Liberty design summit in Vancouver, the Ironic team generally agreed to move the Ironic project to a feature-based release model.[0] This is an informational spec to lay out why we are doing this and how it will work.

Problem description

Ironic currently uses the traditional OpenStack release model.[1] This involves:

- One release every six months.
- A feature freeze prior to cutting this release, typically for a duration of six weeks. Only bug fix work and docs work should be done in this period.
- An optional spec freeze a few weeks prior to the feature freeze.
- After the feature freeze period, a stable branch is forked from master and a release candidate period begins. Critical bugs fixed on master during this period may be backported to the stable branch for the final release.
- After about four weeks of iterating on release candidates, the final version is released. This stable release may receive critical bug fixes for a longer period of 9-15 months.

This model creates a few problems. The primary problem created is that development happens in peaks and valleys. Developers and vendors realize that the feature freeze is coming up, and attempt to merge features quickly before the freeze. This creates a flurry of both patches and reviews, which has the tendency to burn out core reviewers. Then feature freeze begins. Features cannot be landed and development slows, almost to a halt. This typically takes six weeks. Once the next cycle opens for feature development, code starts to trickle through but most folks are working on summit planning and taking a break from the stress of the last release. This essentially causes *10 weeks* of developer downtime *per cycle*, or 20/52 weeks per year.

Proposed change

We should release Ironic (roughly) following the independent release model defined by the OpenStack governance.[2]

Note

As of July 15, 2015 [10], ironic follows the cycle-with-intermediary release model [11]. This model, which better matches this specification, was created by the OpenStack governance after this specification had been approved.

There are a few things of note here.

- We should release Ironic when major features or bug fixes are complete on the master branch. This will be at the discretion of Ironics PTL or a designated release manager.
- We should continue to release a final release every six months, to continue to be part of the coordinated release.[3]
- In lieu of a feature freeze, Ironics stable release should come from a normal Ironic release that happens around the same time that other integrated projects begin the release candidate period. For example, for Liberty[4], Ironic should aim to make a release in the second half of September. This will become equivalent to the stable/liberty branch. Release candidates should be built from this branch and may receive bug fixes. Stable releases should be eligible for receiving backports following the current process.
- Ironic releases should roughly follow SemVer[5] the difference being that the major version should be bumped for significant changes in Ironics functionality or code. Ironic should never do a release without an upgrade path from a prior release, so in traditional SemVer the major version would never change. We should bump minor and patch versions as needed for minor feature and bug fix releases.
- Ironic releases should be published to PyPI.
- Specs should be no longer be targeted to a particular release folks should just work on specs and code continuously. Features will be released as they land. The ironic-specs repo should change to:
  - One approved directory where all specs live initially.
  - Separate $version-implemented directories that house the specs that were implemented in $version of Ironic. Specs are moved to this directory when the work is completed.
  - Create a placeholder in the old location which indicates which release of Ironic the work was completed in, with a link to the new location. This will keep older links from breaking.
- Leading up to all releases, reviewers should honor a soft freeze period of a few days to a week. Code that is risky in terms of breakage should probably not land at this time; quick bug fixes, driver

changes, and other less risky changes should be okay to land in most cases. The PTL or designated release manager must be sure to communicate upcoming releases and these freezes well.

- Ironic may need to decouple from global requirements during Dep Freeze[7]. During OpenStacks feature freeze, the master branch of global-requirements is locked. It should be okay to accept changes to Ironics requirements.txt on the master branch that are blocked in global-requirements by the Dep Freeze, as the stable branch has already been cut by this point. This should only happen on an as-needed basis as the problem arises; not by default. We also may need to temporarily drop the gate-ironic-requirements job during this time. Finally, any patches to Ironic changing requirements.txt should also have a patch to global-requirements with a general looks good from a global-requirements core reviewer.
- Folks have discussed using feature branches for larger chunks of work. While these can be useful, we must be sure to only use them when absolutely necessary as feature branches can carry a large amount of pain. We should prefer feature flags[9] over feature branches where possible.

Alternatives

Continue on the status quo. :(

Data model impact

None.

State Machine Impact

None.

REST API impact

None.

Client (CLI) impact

None. The client will continue to release independently, and likely more often than the server.

RPC API impact

None.

Driver API impact

None.

Nova driver impact

As the Nova driver is released with Nova, it will not change in terms of the way it is released.

Security impact

As only stable releases will receive backports, security bugs in other releases should be fixed and released ASAP. An advisory should be published that encourages users to upgrade to the new release.

Stable branches should continue to receive backports for security bug fixes.

Intermediate releases will not receive backports for security patches. Any security bug in an intermediate release should be fixed and released with the appropriate version bump. Whether the version change is major/minor/patch may depend on what else has landed on master and will be released with the patch.

Other end user impact

End users will get features shipped to them more quickly.

Scalability impact

None.

Performance Impact

None.

Other deployer impact

Deployers will receive changes and features more quickly. Those that do not wish to do so may continue to consume the six-month integrated release.

Developer impact

All the productivity.

Implementation

Assignee(s)

PTL, designated release manager (if one exists), and core reviewers.

Work Items

- Switch to semver.
- Start releasing independently.
- Document the process in the developer docs.[8]

Dependencies

None.

Testing

None.

Upgrades and Backwards Compatibility

This should cause upgrades to be smaller and thus less impactful.

Documentation Impact

We should write a page in our developer docs about the process, including the changes to the specs process.

References

[0] https://etherpad.openstack.org/p/liberty-ironic-scaling-the-dev-team

[1] https://governance.openstack.org/tc/reference/tags/release_at-6mo-cycle-end.html

[2] https://governance.openstack.org/tc/reference/tags/release_independent.html

[3] https://governance.openstack.org/tc/reference/tags/integrated-release.html

[4] https://wiki.openstack.org/wiki/Liberty_Release_Schedule

[5] https://semver.org/

[6] http://lists.openstack.org/pipermail/openstack-dev/2015-May/065211.html

[7] https://wiki.openstack.org/wiki/DepFreeze

[8] https://docs.openstack.org/developer/ironic/

[9] https://en.wikipedia.org/wiki/Feature_toggle

[10] https://review.openstack.org/#/c/202208/

[11] http://governance.openstack.org/reference/tags/release_cycle-with-intermediary.html

5.15.48 iRMC Virtual Media Deploy Driver for Ironic

https://blueprints.launchpad.net/ironic/+spec/irmc-virtualmedia-deploy-driver

The proposal presents the work required to add support for deployment features for FUJITSU PRIMERGY iRMC, integrated Remote Management Controller, Drivers in Ironic.

Problem description

FUJITSU PRIMERGY servers are capable of booting from virtual media, but Ironic lacks a driver which can utilize this for PXE/TFTP less deployment.

Proposed change

Adding new iRMC Drivers, namely iscsi_irmc and agent_irmc, to enable PXE/TFTP less deployment capability to provision PRIMERGY bare metal nodes (having iRMC S4 and beyond) by booting the bare metal node with virtual media using NFS or CIFS from a conductor node to deploy an image.

iRMC virtual media deploy driver is basically same as iLOs. However comparing iRMC deploy driver with iLO deploy driver, the only significant change is the location of virtual media images, specifically deploy ISO image, floppy FAT image and boot ISO image. The location where iRMC deploy driver places the created floppy FAT image and the boot ISO image is on an NFS or CIFS server, while the location where iLO deploy driver creates is on Swift Object Storage Service. The location where iRMC mounts the three images is from an NFS or CIFS server, while the location where iLO mounts is from the http temp-url generated for the Swift object Service. The other parts are common.

Before starting Ironic conductor, however, deployer has to set up operating system properly so that Ironic conductor mounts the NFS or CIFS shared file system on path /remote_image_share_root as default, the path is configurable in the ironic configuration file. This driver checks this mount at start up time.

The NFS or CIFS server can be located anywhere in the network as long as iRMC and Ironic conductor can reach it. The NFS or CIFS server and the network path should be redundant so that a bare metal node wont fail to boot. For this reason, Ironic conductor is not recommended to be used as the NFS or CIFS server for high available environments.

The iRMC deploy module uses python-scciclient package to communicate with ServerView Common Command Interface (SCCI) via HTTP/HTTPS POST protocol.

The details of ServerView Common Command Interface (SCCI) is described in FUJITSU Software ServerView Suite, Remote Management, iRMC S4 - integrated Remote Management Controller

Alternatives

Other drivers such as the following can be used, but there are no drivers that can boot using virtual media.

- iRMC driver for PXE (pxe_irmc) which can boot in either Legacy mode or UEFI. The details of the boot mode control is described in iRMC Management Driver for Ironic.
- IPMI driver for PXE (pxe_ipmitool) which can boot only in Legacy mode.

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Security impact

- Admin credentials will be stored unencrypted in the Ironic configuration file, and the DB which will be visible in the driver_info field of the node when a node-show is issued. But only the ironic admin user will have access to the Ironic configuration file and the DB.

- NFS and CIFS have very similar functionality, but have different security model. NFS server authenticates NFS client based on NFS client host IP address, while CIFS server authenticates CIFS client based on user identity. For this reason, this driver deals with CIFS as default, and NFS as alternative. Advanced deployer could use NFSv4 and Kerberos to authenticate NFS client based on user identity. Detail information is available in each linux distribution manual ([1],[2])

Other end user impact

None

Scalability impact

NFS or CIFS server would become performance bottleneck in case of managing a large number of bare metal nodes such as more than 1,000 nodes. In that case, the load needs to be distributed and balanced among multiple of NFS or CIFS server settings ([3],[4],[5])

Performance Impact

None

Other deployer impact

- The following parameters are required in the [irmc] section of the ironic configuration file which is typically located at /etc/ironic/ironic.conf in addition to the parameters defined in iRMC Power Driver for Ironic
  - remote_image_share_root: Ironic compute nodes NFS or CIFS root path (string value). The default value is /remote_image_share_root.
  - remote_image_server: IP of remote image server
  - remote_image_share_type: The share type (NFS or CIFS) of virtual media. The default value is CIFS.
  - remote_image_share_name: The share name of remote_image_server. The default value is share.
  - remote_image_user_name: user name of remote_image_server
  - remote_image_user_password: password of remote_image_user_name
  - remote_image_user_domain: domain name of remote_image_user_name. The default value is .
- The following driver_info field is required to support iRMC virtual media in addition to the fields defined in iRMC Power Driver for Ironic.
  - irmc_deploy_iso: deploy ISO image which is either a file name relative to remote_image_share_root, Glance UUID, Glance URL or Image Service URL.
- The following instance_info field is optional.

[1] https://help.ubuntu.com/community/NFSv4Howto
[2] http://docs.fedoraproject.org/en-US/Fedora/17/html/FreeIPA_Guide/kerb-nfs.html
[3] https://help.ubuntu.com/community/HighlyAvailableNFS
[4] https://wiki.samba.org/index.php/Samba_CTDB_GPFS_Cluster_HowTo
[5] https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Cluster_Administration/ch-clustered-samba-CA.html

  - irmc_boot_iso: boot ISO image file name relative to remote_image_share_root, Glance UUID, Glance URL or Image Service URL. If it is not specified, the boot ISO is created automatically from registered images in Glance.

- In order to use iRMC virtual media deploy driver, iRMC S4 and beyond with iRMC a valid license is required. Deployer is notified by error message if the iRMC version and/or the license is not valid.
- In order to deploy and boot ISO image via virtual media, an NFS or CIFS server is required. The NFS or CIFS server has to be reachable from both iRMC and Ironic conductor.

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
Naohiro Tamura (naohirot)

Other contributors:
None

Work Items

- Add iRMC Drivers (iscsi_irmc, agent_irmc)
- Implement iRMC virtual media deploy module for the iRMC Drivers by reusing and refactoring some part of the code from the current iLO deploy driver.

Dependencies

- This feature requires iRMC S4 and beyond that is at least BX S4 or RX S8 generation of FUJITSU PRIMERGY servers.
- This feature uses python-scciclient package.
- This feature depends on iRMC Power Driver for Ironic and iRMC Management Driver for Ironic.

Testing

- Unit Tests
- Fujitsu plans Third-party CI Tests

Upgrades and Backwards Compatibility

None

Documentation Impact

The required driver_info fields and [irmc] section parameters in the ironic configuration file need be included in the documentation to instruct operators how to use Ironic with iRMC.

References

- FUJITSU Software ServerView Suite, Remote Management, iRMC S4 - integrated Remote Management Controller
- iRMC Power Driver for Ironic
- iRMC Management Driver for Ironic
- python-scciclient package
- iLO Virtual Media iSCSI Deploy Driver
- iLO IPA Deploy Driver
- Automate UEFI-BIOS Iso Creation
- Support for non-glance image references

5.15.49 Open CloudServer (OCS) power driver

https://blueprints.launchpad.net/ironic/+spec/msft-ocs-power-driver

This blueprint adds support for the Open CloudServer (OCS) v2.0 power interface in Ironic. The OCS design and specs have been contributed by Microsoft to the Open Compute project.

Problem description

The OCS chassis system includes a chassis manager server which exposes a REST API to manage the individual blades, replacing traditional protocols like IPMI. The REST API service itself is open source (Apache 2 license).

In order to be able to execute power and management actions on OCS blades, the corresponding interfaces need to be implemented.

Furthermore, the OCS REST API also supports a serial console interface for individual blades that can be supported in Ironic.

Proposed change

The proposed implementation consists of a driver implementation. A client will be provided to abstract the OCS service REST API calls, which in turn can be referenced by the power, management and console interfaces.

Both UEFI and legacy BIOS boot modes are supported and can be specified by the user as part of the properties/capabilities.

Driver properties that can be specified by the user:

msftocs_base_url

Base url of the OCS chassis manager REST API, e.g.: http://10.0.0.1:8000. Required.

msftocs_blade_id

Blade id, must be a number between 1 and the maximum number of blades available in the chassis. In the current configuration OCS chassis have a maximum of 24 or 48 blades. Required.

msftocs_username

Username to access the chassis manager REST API. Required.

msftocs_password

Password to access the chassis manager REST API. Required.

Alternatives

No alternatives are available for the OCS case.

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Security impact

The interaction between Ironic and the OCS chassis manager involves REST API calls, using HTTP basic authentication and potentially NTLM authentication in the future.

The HTTP credentials are provided by the user as part of the driver properties and need to be passed to the REST API service. It is highly recommended to employ HTTPS for transport encryption in any production environment.

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

None

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
<alexpilotti>

Other contributors:
<atuvenie>

Work Items

- Power and management interfaces
- Console interface

Dependencies

None

Testing

Potential continuous integration system integrated with Gerrit / Zuul. The challenge is that a non trivial amount of OCS resources is required for this purpose.

Upgrades and Backwards Compatibility

None

Documentation Impact

The driver should be documented in a way similar to other Ironic drivers under http://docs.openstack.org/developer/ironic/index.html

References

- OCS design and specs: http://www.opencompute.org/wiki/Server/SpecsAndDesigns
- Chassis Manager sources: https://github.com/MSOpenTech/ChassisManager

5.15.50 Boot Interface in Ironic

https://blueprints.launchpad.net/ironic/+spec/new-boot-interface

This spec talks about refactoring the boot logic out of the current Ironic deploy drivers into a new boot interface.

Problem description

Current we have a DeployInterface in Ironic. All the current implementations of this interface are responsible for two things:

- Booting the bare metal node - both deploy ramdisk and the deployed instance.
- Actual deployment of an image on a bare metal node.

These are two separate functions and therefore should be abstracted separately. This makes it easy to mix and match various boot mechanisms (like PXE, iPXE, virtual media) with various deploy mechanisms (iscsi deploy, agent deploy and various other deploy mechanisms which may be possible like deploying with torrent, multicast, etc in which discussions have been initiated) without duplicating code.

Proposed change

- A new `BootInterface` needs to be added. The interface will recommend the following methods for the implementer:

```
@six.add_metaclass(abc.ABCMeta)
class BootInterface(object):
  """Interface for inspection-related actions."""

  @abc.abstractmethod
  def get_properties(self):
      """Return the properties of the interface.

      :returns: dictionary of <property name>:<property description>
     """

  @abc.abstractmethod
  def validate(self, task):
      """Validate the driver-specific info for booting.

      This method validates the driver-specific info for booting a ramdisk
      and an instance on the node.  If invalid, raises an exception;
      otherwise returns None.

      :param task: a task from TaskManager.
      :raises: InvalidParameterValue
      :raises: MissingParameterValue
      """

  @abc.abstractmethod
  def prepare_ramdisk(self, task, ramdisk_params):
      """Prepares the boot of Ironic ramdisk.
```

(continues on next page)

(continued from previous page)

```
        This method prepares the boot of the deploy ramdisk after reading
        relevant information from the node's database.

        :param task: a task from TaskManager.
        :param ramdisk_params: the options to be passed to the ironic␣
↪ramdisk.
            Different interfaces might want to boot the ramdisk in different
            ways by passing parameters to them.  For example,
            * When DIB ramdisk is booted to deploy a node, it takes the
              parameters iscsi_target_iqn, deployment_id, ironic_api_url,␣
↪etc.
            * When Agent ramdisk is booted to deploy a node, it takes the
              parameters ipa-driver-name, ipa-api-url, root_device, etc.
            Other interfaces can make use of ramdisk_params to pass such
            information.  Different implementations of boot interface will
            have different ways of passing parameters to the ramdisk.
        """

    @abc.abstractmethod
    def clean_up_ramdisk(self, task):
        """Tears down the boot of Ironic ramdisk.

        This method tears down the boot of the deploy ramdisk after reading
        relevant information from the node's database.

        :param task: a task from TaskManager.
        """

    @abc.abstractmethod
    def prepare_instance(self, task):
        """Prepares the boot of instance.

        This method prepares the boot of the instance after reading
        relevant information from the node's database.

        :param task: a task from TaskManager.
        """

    @abc.abstractmethod
    def clean_up_instance(self, task):
        """Tears down the boot of instance.

        This method tears down the boot of the instance after reading
        relevant information from the node's database.

        :param task: a task from TaskManager.
        """
```

- The following new implementations of `BootInterface` will be created.
  - `pxe.PXEBoot` - Booting a bare metal node using PXE

  - `ipxe.IPXEBoot` - Booting a bare metal node using iPXE
  - `ilo.boot.IloVirtualMediaBoot` - Booting a bare metal node using iLO Virtual Media.

> **Note**
>
> Even though IPXEBoot and PXEBoot are in same deploy driver currently, the steps for preparing a bare metal to boot from PXE and iPXE are different (even though they share some common code). We will refactor both of them as separate boot interfaces. The Kilo behaviour of using only either of PXE or iPXE at same time will be retained - drivers will instantiate pxe.PXEBoot or ipxe.IPXEBoot depending on CONF.pxe.ipxe_enabled.

- The code for the above implementations of `BootInterface` will be taken from `pxe.PXEDeploy`, `agent.AgentDeploy`, `ilo.IloVirtualMediaIscsiDeploy` and `ilo.IloVirtualMediaAgentDeploy`. These implementations of `DeployInterface` will be freed of any logic dealing with booting of bare metal node.
- `pxe.PXEDeploy` will be refactored into `pxe.PXEBoot` and `iscsi_deploy.ISCSIDeploy`.
- Each driver will mention what is the `BootInterface` implementation that it wishes to instantiate. For example, the `pxe_ipmitool` driver will look like the following:

```
class PXEAndIPMIToolDriver(base.BaseDriver):
  """PXE + IPMITool driver"""

  def __init__(self):
    self.power = ipmitool.IPMIPower()
    self.console = ipmitool.IPMIShellinaboxConsole()
    self.boot = pxe.PXEBoot()
    self.deploy = iscsi_deploy.ISCSIDeploy()
    self.management = ipmitool.IPMIManagement()
    self.vendor = pxe.VendorPassthru()
    self.inspect = discoverd.DiscoverdInspect.create_if_enabled(
        'PXEAndIPMIToolDriver')
```

> **Note**
>
> It might make sense to rename the drivers to include the boot interface as well as deploy interface after this is implemented. As such, this requires a better thought out process to rename the drivers, address issues of backward compatibility, etc. Hence it is out of scope of this spec. That can be addressed later after this is implemented.

Alternatives

We can continue to keep the boot and deploy logic together but this will lead to code duplications and unnecessary refactorings when additional deploy mechanisms and boot mechanisms are added in the future.

Data model impact

None.

State Machine Impact

None.

REST API impact

None.

RPC API impact

None.

Client (CLI) impact

None.

Driver API impact

This adds the a new `BootInterface` (as described above) which driver writers may use with the deploy drivers. `BootInterface` is not a mandatory interface.

Nova driver impact

None.

Security impact

None.

Other end user impact

None.

Scalability impact

None.

Performance Impact

None.

Other deployer impact

None.

Developer impact

New driver developers adding new deploy mechanisms in Ironic will be encouraged to separate boot and deploy logic so that it can reused easily.

Implementation

Assignee(s)

rameshg87

Work Items

- Add new boot interface
- Create `pxe.PXEBoot`, `ipxe.IPXEBoot` and refactor `pxe.PXEDeploy` into `iscsi_deploy.ISCSIDeploy` to make use of these boot interfaces.
- Refactor `agent.AgentDeploy` to use new `pxe.PXEBoot` and `ipxe.IPXEBoot` (Yes, we are adding iPXE support for agent deploy).
- Create `ilo.boot.IloVirtualMediaBoot`, and refactor `IloVirtualMediaIscsiDriver`, `IloVirtualMediaAgentDriver` to make use of the new boot interface.

Dependencies

None.

Testing

Unit tests will be updated for the new interfaces. Since this change doesnt add any new functionality, the current upstream CI testing should be enough.

Upgrades and Backwards Compatibility

This doesnt break out-of-tree deploy drivers. Still it will be possible to implement deploy drivers for provisioning bare metal nodes without a boot interface- i.e without separate boot and deploy interfaces. This is because the conductor will still be using all the published interfaces of `DeployInterface` for deploying a bare metal node.

This change proposes the addition of new optional boot interface which can be used as a helper for `DeployInterface` and refactors all upstream deploy drivers to follow this logic.

Documentation Impact

Changes to the existing interface will be documented. Also, new developer documentation will be updated to encourage splitting deploy logic into separate boot and deploy interfaces.

References

Not according to this spec, but a POC how it will look like: * https://review.openstack.org/#/c/166512/ * https://review.openstack.org/#/c/166513/ * https://review.openstack.org/#/c/166521/

5.15.51 Make ilo drivers standalone in ironic by removing swift dependency

https://blueprints.launchpad.net/ironic/+spec/remove-swift-dependency-for-ilo-drivers

This spec proposes to remove hard dependency on swift for ilo virtual media drivers. There are standalone use-cases of Ironic (like bifrost) which are capable of deploying nodes without requiring other Openstack services. With this the ilo virtual media drivers can also work standalone similar to other drivers in ironic.

Problem description

Today the ilo drivers (iscsi_ilo and agent_ilo) require swift to host the images like boot_iso (from which the instance boots) and floppy images (used for passing parameters to deploy ramdisk).

Proposed change

The ilo drivers (iscsi_ilo and agent_ilo) can use a web server to host the images required. Swift would be used as a default backend to host boot ISO and floppy images for these drivers.

- The automatic boot_iso created (as required by iscsi_ilo for booting up the instance) and floppy images (as required by both agent_ilo and iscsi_ilo for passing parameters to deploy ramdisk) during deploy process, will be hosted on swift or http web server as per the config variable under *[ilo]* in ironic.conf:

  ```
  use_http_web_server_for_images=True
  ```

  The default value would be *False* and will default to use swift.

- User needs to manually configure the webserver, and add the config options in ironic.conf under *deploy* as:

  ```
  http_server_root = /opt/stack/ironic/data/httpboot
  http_server_url = http://10.10.1.30/httpboot
  ```

  Since the same config variables exists under *[pxe]* and are required by ilo drivers to be able to run standalone, we can deprecate the same in *[pxe]* and move it under *[deploy]*. The above values to the config variables are just an example. They will continue to have the default value as the current config variables `http_url` and `http_root`.

- To add the functionality of take_over() to ilo drivers. This is to enable a case when the conductor node goes down and other conductor takes over the baremetal. The take_over() will be implemented to do regenerate in following scenarios:
  - for wait_call_back state: implement regeneration of floppy images.
  - for active state : implement regeneration of boot ISO.

Alternatives

None.

Data model impact

None.

State Machine Impact

None.

REST API impact

None.

Client (CLI) impact

None.

RPC API impact

None.

Driver API impact

None.

Nova driver impact

None.

Security impact

None.

Other end user impact

None.

Scalability impact

None.

Performance Impact

None.

Other deployer impact

The http web server configuration is out of scope of ironic but it should be configured on every conductor node. User needs to manually configure the web server, and add the config options in ironic.conf under *deploy* as:

```
http_server_root = /opt/stack/ironic/data/httpboot
http_server_url = http://10.10.1.30/httpboot
```

Since the same config variables exists under *[pxe]* and is required by ilo drivers to be able to run standalone, we can deprecate the same in *[pxe]* and move it under *[deploy]*.

Developer impact

None.

Implementation

Assignee(s)

Primary assignee:
agarwalnisha1980

Work Items

- To modify ipxe to use config variables from *[deploy]* section.
- To enable support for using webserver in ilo drivers.
- To implement take_over() for ilo_drivers for `use_web_server=True`

Dependencies

None.

Testing

- Mocked unit tests would be added.
- Functional tests would be done to ensure that the deploy works fine for ilo drivers with swift as backend or http webserver as backend.

Upgrades and Backwards Compatibility

The ilo drivers will continue to work with swift as backend. The iPXE code will continue to work for config options in either of the section.

Documentation Impact

It is required to be documented.

References

None.

5.15.52 Determinable supported boot device list

https://blueprints.launchpad.net/ironic/+spec/supported-boot-device-list

This blueprint proposes to add the facility to return a determinable list of supported boot devices that is specific to the node being queried.

Problem description

Current driver interface get_supported_boot_devices() does not provide for a means to access the node being queried, as a result a determinable list of supported boot devices that are specific to this node cannot be calculated.

Example usage, if a node uses an architecture other than x86 e.g. SPARC, then PXE boot is not supported, and therefore should not be contained in the returned list. SPARC architecture supports WANBOOT and this should be returned. However to determine this the nodes property cpu_arch needs to be accessed.

Proposed change

Add task parameter to get_supported_boot_devices() interface.

By providing a task parameter to the get_supported_boot_devices() interface a driver can then access the node specific to this query e.g. task.node, and from there can look at a nodes properties and correctly determine the list of supported boot devices to return.

This change would be backwards compatible, using inspect to determine if a specific driver as implemented the task parameter or not, and showing a deprecation warning if task parameter has not been implemented.

This deprecation of get_suported_boot_devices() without task parameter will be done in the next release, after which get_supported_boot_devices() without the task parameter will not be supported.

Alternatives

The only method to access the specific node for which a get_supported_boot_devices request is being made is via passing in the task argument.

Alternative 1: Restrict all drivers to only be able to register nodes of a specific architecture. Then the underlying driver API for get_supported_boot_devices() would just assume that this node must be of a specific architecture and always return a static list for that architecture.

This approach is limiting and would result in an unnecessary increase in the number of drivers available for ironic, and a unnecessary potential duplication of code.

Alternative 2: Utilize set_boot_device(task, device), this method in most cases will validate if a device is supported for this node. A task parameter is being passed in here, and thus the node can be accessed to determine if device is supported for this specific node. The issue here is that it would have to be called a number of times to determine a list of supported devices, and each successful call would result in an IPMI call to actually set the boot device for the node which would be unacceptable and inefficient.

This approach is not complete and does not solve the scenario where get_supported_boot_devices() is called via the ironic CLI. This would still result in a potentially incorrect list of devices being returned to the user.

Alternative 3: This is more of an addendum to the proposed solution. Add a new essential property for a node called supported_boot_devices. This property would be populated during node inspection, and could then be queried to determine the list of supported boot devices.

Access to the node however would still be required, so passing of the task parameter to get_supported_boot_devices would still be required.

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

RPC API impact

None

Driver API impact

get_supported_device_list() is a member of the standard Management interface.

This change would effect all drivers but would be implemented so that its backward compatible, using inspect to determine if a driver supports the task argument or not. If not a deprecation warning would be shown.

Nova driver impact

None

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

None

Developer impact

All in-tree driver and unit tests will be amended to include the new task parameter. Out of tree drivers will see a deprecation warning until they implement the new task parameter.

Implementation

Assignee(s)

Primary assignee:
mattkeenan

Work Items

- Add task to get_supported_devices_list() in base driver
- Add new management property definition to check if driver supports task parameter or not.
- Update all in tree drivers adding task parameter
- Update all unit tests affected by change

Dependencies

None

Testing

Will update all affected unit tests

Upgrades and Backwards Compatibility

Backwards compatibility is achieved by using pythons inspect module to determine if a drivers get_supported_device_list() implementation includes a task parameter or not. If not it will be called without the task parameter and a deprecation warning will be shown.

Documentation Impact

None

References

Bug: https://bugs.launchpad.net/ironic/+bug/1391598 Review: https://review.openstack.org/#/c/188466

5.15.53 UEFI Secure Boot support for pxe_iLO driver

Include the URL of your launchpad blueprint:

https://blueprints.launchpad.net/ironic/+spec/uefi-secure-boot-pxe-ilo

As part of Kilo release UEFI secure boot support was enabled for all the iLO drivers except pxe_ilo. It is important to have this feature supported for pxe_ilo driver so that security sensitive users of pxe_ilo driver could deploy more securely using Secure Boot feature of the UEFI. This spec proposes UEFI Secure Boot support in baremetal provisioning for pxe_ilo driver.

Problem description

Secure Boot is part of the UEFI specification (http://www.uefi.org). It helps to make sure that node boots using only software that is trusted by Admin/End user.

Secure Boot is different from TPM (Trusted Platform Module). TPM is a standard for a secure cryptoprocessor, which is dedicated microprocessor designed to secure hardware by integrating cryptographic keys

into devices. Secure Boot is part of UEFI specification, which can secure the boot process by preventing the loading of drivers or OS loaders that are not signed with an acceptable digital signature.

When the node starts with secure boot enabled, system firmware checks the signature of each piece of boot software, including firmware drivers (Option ROMs), boot loaders and the operating system. If the signatures are good, the node boots, and the firmware gives control to the operating system.

The Admin and End users having security sensitivity with respect to baremetal provisioning owing to the workloads they intend to run on the provisioned nodes would be interested in using secure boot provided by UEFI.

Once secure boot is enabled for a node, it cannot boot using unsigned boot images. Hence it is important to use signed bootloaders and kernel if node were to be booted using secure boot.

This feature has been enabled for iscsi_ilo and agent_ilo driver during Kilo release. It needs to be enabled for pxe_ilo driver. This needs pxe_ilo driver should support signed UEFI bootloader for the nodes to boot in the UEFI secure boot environment.

Proposed change

This spec proposes to support UEFI secure boot for pxe_ilo driver and grub2 as an alternate bootloader for UEFI deploy for PXE drivers.

Preparing the environment

- The operator informs the Ironic using the `capabilities` property of the node. The operator may add a new capability `secure_boot=true` in `capabilities` within `properties` of that node. This is an optional property that can be used if node needs to be provisioned for secure boot. By default the behavior would be as if this property is set to false. The inspection feature in iLO drivers can auto discover secure boot capability of the node and create node capability into that node object.
- If the user has `secure_boot` capability set in the flavor, pxe_ilo has ability to change the boot mode to UEFI and prepare the node for the secure boot on the fly using proliantutils library calls.
- Even if the `secure_boot` capability is set to `true` in the nodes `properties/capabilities`, node can be used for normal non-secure boot deployments. Driver would use the `secure_boot` capability information from the nodes `instance_info` field to provision node for UEFI secure boot.

Preparing flavor for secure boot

- The `extra_specs` field in the nova flavor should be used to indicate secure boot. User will need to create a flavor by adding capabilities:secure_boot=true to it.
- iLO driver will not do secure boot if secure_boot capability flavor is not present or set to False. Nova scheduler will use secure_boot capability as one of the node selection criteria if secure_boot is present in extra_spec. If secure_boot is not present in extra_spec then Nova scheduler will not consider secure_boot capability as a node selection criteria.
- Ironic virt Driver will pass the flavor capability information to the driver as part of instance_info. Having capability information as part of instance_info would help driver in preparing and decommissioning the node appropriately. With respect to secure boot feature, this information would be used by pxe_ilo driver for:-

  - During provisioning, driver can turn on the secure boot capability to validate signatures of bootloaders and kernel.
  - During teardown, secure boot mode would be disabled on the node.

Preparing bootloader and deploy images

To support UEFI secure boot for pxe_ilo driver, pxe driver for Ironic should support signed UEFI bootloader. Currently elilo is the default UEFI bootloader for all pxe drivers. Not all major linux distros ship signed elilo bootloader. They ship signed grub2 bootloader.

Enabling grub2 bootloader requires steps similar to elilo. Steps are:-

- Copy signed shim and grub2 bootloader files into tftproot directory as bootx64.efi and grubx64.efi respectively .
- Create a master grub.cfg file under /tftpboot/grub
- Contents of master grub.cfg would look something like this. set default=master set timeout=5 set hidden_timeout_quiet=false

  menuentry master { configfile /tftpboot/$net_default_ip.conf }

  This master grub.cfg gets loaded first during PXE boot. It tells grub to refer to the node specific config file in tftproot directory configured for PXE. The name of config file is coined using DHCP IP address that would be allocated to the node. This is to ensure that multiple grub.cfg files could be created for parallel deploys. The contents of $net_default_ip.conf is dynamically filled by PXE driver using grub template file.

Ironic needs to support grub2 as an alternate UEFI bootloader for following reasons:-

- No active development happening on elilo
- All major linux distributions are supporting grub2 as a default UEFI bootloader.
- All major linux distributions provide signed grub2 bootloader which could be used in UEFI secure boot deploy with distro supplied cloud images. Otherwise users would need to build their own signed images for secure boot deploy.
- signed grub2 can be used for normal UEFI deploys as well.

All major linux distros ship their self signed grub2 and also provide Microsoft UEFI CA signed shim bootloader. The shim bootloader contains the UEFI signature of respective distros.

When node boots up using pxe, it loads Microsoft signed shim boot loader which in turn loads the distro signed grub2. Distro signed grub2 can validate and load the distro kernel. Shim bootloader is required as it is signed using Microsoft UEFI CA signature and recognizes corresponding linux vendors certificate as a valid certificate. Secure boot enabled HP Proliant UEFI systems are pre-loaded with Microsoft UEFI CA signatures. User signed images can be supported but user need to manually configure their keys to HP Proliant system ROM database using Proliant tools.

User can configure grub2 as a bootloader by changing the following existing variables in /etc/ironic/ironic.conf under pxe section: uefi_pxe_config_template uefi_pxe_bootfile_name

Alternatives

Add support for signed grub2 as a default UEFI bootloader in Ironic. But such a change would have backward compatibility impact.

Data model impact

None

State Machine Impact

None

REST API impact

None

RPC API impact

None

Client (CLI) impact

None

Driver API impact

None

Nova driver impact

None

Security impact

This enhances security. Only correctly signed firmware, bootloader and OS can be booted. It provides users with the opportunity to run the software of their choice in the most secure manner.

Other end user impact

Users need to use properly signed deploy and boot components. Currently pxe_ilo driver would support deploy and boot images having shim and grub2 signed by Linux OS vendors. If user wants to use custom signed images, then he would need to manually configure their keys to UEFI using HP Proliant tools. If user were to use an unsigned image for deploy with flavor requesting UEFI secure boot, then deploy process would go through successfully, but final boot into instance image would fail. The signature validation of unsigned components would fail resulting in the failure of boot process. The appropriate boot failure message would get displayed on Nodes console.

Scalability impact

None

Performance Impact

There is no performance impact due to signature validation in secure boot.

Other deployer impact

User can deploy only signed images with UEFI secure boot enabled.

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
Shivanand Tendulker (stendulker@gmail.com)

Work Items

1. Add support for grub2/shim as a alternate UEFI bootloaders for Ironic pxe driver.
2. Implement secure boot for pxe_ilo driver.

Dependencies

Signed user images. The necessary DiskImageBuilder changes has been done to build signed Ubuntu and Fedora images.

Testing

Unit tests would be added for all newly added code.

Upgrades and Backwards Compatibility

None. grub2 would be alternate bootloader, which user can use only if it needs UEFI secure boot functionality.

Documentation Impact

Newly added functionality would be appropriately documented.

References

1. UEFI specification http://www.uefi.org
2. Proliantutils module - https://pypi.python.org/pypi/proliantutils
3. HP UEFI System Utilities User Guide - http://www.hp.com/ctg/Manual/c04398276.pdf
4. Secure Boot for Linux on HP Proliant servers http://h20195.www2.hp.com/V2/getpdf.aspx/4AA5-4496ENW.pdf

5.15.54 Wake-On-Lan (WOL) power driver

https://blueprints.launchpad.net/ironic/+spec/wol-power-driver

This blueprint adds support for Wake-On-Lan (WOL) power interface in Ironic.

Problem description

Wake-On-Lan is a standard that allows a computer to be powered on by a network message. This is widely available and doesnt require any fancy hardware to work with. This is useful for users that want to try Ironic with real bare metal instead of virtual machines and only have some old PCs around.

Proposed change

The proposed implementation consists of a driver power interface implementation that will create and send Wake-On-Lan magic packets for each port (MAC addresses) registered on the node resource in Ironic.

Its important to note that Wake-On-Lan is only capable of powering **on** the machine. After the machine is unprovisioned it needs to powered off manually.

Driver properties that can be specified by the user::

wol_host: Broadcast IP address to send the magic packets. Defaults to `255.255.255.255`.

wol_port: Destination port to send the magic packets; defaults to 9.

When powering **off** is called we are just going to log a message saying the operation isnt supported by the driver and require manual intervention to be performed.

When **reboot** is called the driver will try to power **on** the machine.

When getting the power state of the node, the driver will rely on whatever is in the Ironic database and return it, since we dont have any reliable way of knowing the current power state of the machine.

Alternatives

A alternative would be relying on some external mechanism to power control the nodes. Such as `iBoot` which Ironic already have a driver for. But for that the user will need to spend some money (~200 USD) to buy an `iBoot` device.

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Security impact

This spec only covers the bits of sending a magic packet **without** the `SecureOn` feature.

`SecureOn` allows a client to append a password to the magic packet so NICs that support the feature will check prior to powering on the machine and if the MAC address + password are correct only then the system is awake . This is good against `brute force attacks`, but will be left for future work.

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

None

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
lucasagomes

Other contributors:
None

Work Items

- Write the Wake-On-Lan power interface
- Write unittests

Dependencies

None

Testing

Unittests will be added as part of the work.

Upgrades and Backwards Compatibility

None

Documentation Impact

The driver should be documented under http://docs.openstack.org/developer/ironic/index.html. The documentation will also be clear about the use of this driver, this is a testing driver and not meant for production use.

References

- Wake-On-Lan: http://en.wikipedia.org/wiki/Wake-on-LAN

5.15.55 AMT PXE Driver

This spec was implemented in the Kilo cycle.

See *AMT PXE Driver*.

5.15.56 Ironic Microversions

This spec was implemented in the Kilo cycle.

See *Ironic Microversions*.

5.15.57 automate-uefi-bios-iso-creation

This spec was implemented in the Kilo cycle.

See *automate-uefi-bios-iso-creation*.

5.15.58 Driver Internal Info

This spec was implemented in the Kilo cycle.

See *Driver Internal Info*.

5.15.59 Allow drivers to have their own periodic tasks

This spec was implemented in the Kilo cycle.

See *Allow drivers to have their own periodic tasks*.

5.15.60 Expose configdrive to instances

This spec was implemented in the Kilo cycle.

See *Expose configdrive to instances*.

5.15.61 Extend Vendor Passthru

This spec was implemented in the Kilo cycle.

See *Extend Vendor Passthru*.

5.15.62 Implement Cleaning Operations for iLO drivers

This spec was implemented in the Kilo cycle.

See *Implement Cleaning Operations for iLO drivers*.

5.15.63 iLO Management Interface

This spec was implemented in the Kilo cycle.

See *iLO Management Interface*.

5.15.64 Discover node properties and capabilities for iLO drivers

This spec was implemented in the Kilo cycle.

See *Discover node properties and capabilities for iLO drivers*.

5.15.65 Implement Cleaning States

This spec was implemented in the Kilo cycle.

See *Implement Cleaning States*.

5.15.66 In-band hardware properties introspection via ironic-discoverd

This spec was implemented in the Kilo cycle.

See *In-band hardware properties introspection via ironic-discoverd*.

5.15.67 iRMC Management Driver for Ironic

This spec was implemented in the Kilo cycle.

See *iRMC Management Driver for Ironic*.

5.15.68 iRMC Power Driver for Ironic

This spec was implemented in the Kilo cycle.

See *iRMC Power Driver for Ironic*.

5.15.69 Discover node properties with node-set-provision-state

This spec was implemented in the Kilo cycle.

See *Discover node properties with node-set-provision-state*.

5.15.70 Add support for VirtualBox through WebService

This spec was implemented in the Kilo cycle.

See *Add support for VirtualBox through WebService*.

5.15.71 Local boot support with partition images

This spec was implemented in the Kilo cycle.

See *Local boot support with partition images*.

5.15.72 Ironic Logical Names

This spec was implemented in the Kilo cycle.

See *Ironic Logical Names*.

5.15.73 Add maintenance reason field

This spec was implemented in the Kilo cycle.

See *Add maintenance reason field*.

5.15.74 A proposal for the New Ironic State Machine.

This spec was implemented in the Kilo cycle.

See *A proposal for the New Ironic State Machine.*.

5.15.75 Support for non-glance image references

This spec was implemented in the Kilo cycle.

See *Support for non-glance image references*.

5.15.76 Root device hints

This spec was implemented in the Kilo cycle.

See *Root device hints*.

5.15.77 Seamicro Serial Console

This spec was implemented in the Kilo cycle.

See *Seamicro Serial Console*.

5.15.78 UEFI Secure Boot support for iLO drivers

This spec was implemented in the Kilo cycle.

See *UEFI Secure Boot support for iLO drivers*.

5.15.79 Whole Disk Image Support

This spec was implemented in the Kilo cycle.

See *Whole Disk Image Support*.

5.15.80 Add instance_info field to Node model

This spec was implemented in the Juno cycle.

See *Add instance_info field to Node model*.

5.15.81 Add Support for Retry on NodeLocked Exceptions

This spec was implemented in the Juno cycle.

See *Add Support for Retry on NodeLocked Exceptions*.

5.15.82 Agent Driver

This spec was implemented in the Juno cycle.

See *Agent Driver*.

5.15.83 Mechanism to cleanup all ImageCaches

This spec was implemented in the Juno cycle.

See *Mechanism to cleanup all ImageCaches*.

5.15.84 More robust device status checking with fuser

This spec was implemented in the Juno cycle.

See *More robust device status checking with fuser*.

5.15.85 DRAC Management driver for Ironic

This spec was implemented in the Juno cycle.

See *DRAC Management driver for Ironic*.

5.15.86 DRAC Power Driver for Ironic

This spec was implemented in the Juno cycle.

See *DRAC Power Driver for Ironic*.

5.15.87 Enabling IPMI double bridge support

This spec was implemented in the Juno cycle.

See *Enabling IPMI double bridge support*.

5.15.88 API to Get driver_info Properties

This spec was implemented in the Juno cycle.

See *API to Get driver_info Properties*.

5.15.89 iLO IPA Deploy Driver

This spec was implemented in the Juno cycle.

See *iLO IPA Deploy Driver*.

5.15.90 iPXE boot

This spec was implemented in the Juno cycle.

See *iPXE boot*.

5.15.91 iLO Power Driver for Ironic

This spec was implemented in the Juno cycle.

See *iLO Power Driver for Ironic*.

5.15.92 iLO Virtual Media iSCSI Deploy Driver

This spec was implemented in the Juno cycle.

See *iLO Virtual Media iSCSI Deploy Driver*.

5.15.93 Power driver for SNMP-enabled smart PDUs

This spec was implemented in the Juno cycle.

See *Power driver for SNMP-enabled smart PDUs*.

5.15.94 New driver ManagementInterface

This spec was implemented in the Juno cycle.

See *New driver ManagementInterface*.

5.15.95 Send Sensor Data to Ceilometer

This spec was implemented in the Juno cycle.

See *Send Sensor Data to Ceilometer*.

5.15.96 Support external DHCP providers

This spec was implemented in the Juno cycle.

See *Support external DHCP providers*.

5.15.97 Swift Temporary URLs

This spec was implemented in the Juno cycle.

See *Swift Temporary URLs*.

5.15.98 UEFI support for Ironic deploy drivers

This spec was implemented in the Juno cycle.

See *UEFI support for Ironic deploy drivers*.

5.15.99 Ironic L3 based deployment

https://storyboard.openstack.org/#!/story/1749193

Ironic relies on L2 network for IP configuration (and optionally PXE booting) during provisioning baremetal nodes. This imposes limitations on Multi-rack deployments and remote site deployments like Edge Cloud etc. Besides that, lossy provisioning network may slow down or fail some deployments.

This specification proposes relying upon Virtual Media boot capability of modern BMCs to deliver boot image, static network configuration, node identification information and more to the node of choice reliably and securely.

Proposed way of booting the node should fully eliminate the need for DHCP and PXE thus enabling deployment over purely L3 network.

Problem description

It is not always easy to provide Layer-2 connectivity between Ironic conductor node and target nodes.

Example use cases:

- Multi Rack deployment: Inter-Rack L2 switching beyond ToR is a challenge with DHCP-relay, as this adds requirement of either having ToR with DHCP-relay, or having a system or VM listening on the remote subnet to relay or proxy DHCP requests.
- Remote site targets: It would be a great idea to deploy Servers in different remote sites from a central location. With the current setup of L2 dependency we need to provide VPN L2 tunnel between Ironic Conductor and remote target server.

  Example: Edge cloud servers or Distributed VNF Flexi Zone Controllers[1].
- Lossy, overloaded provisioning network may cause DHCP or TFTP packet drop slowing down or failing the entire node deployment.

Most of the hardware offerings on the market support booting off virtual media device. While Ironic can boot the nodes over virtual media, its present workflow still relies on DHCP for booted OS to obtain IP stack configuration. This dependency on DHCP can be eliminated if proposed change is implemented.

Proposed change

Deploying a node without DHCP involves solving three crucial problems:

1. Securely delivering boot image to the right node
2. Gathering node configuration information from cloud or site-specific infrastructure
3. Provisioning node configuration (as well as authentication, identification etc) information to the ramdisk operating system running on the node

Virtual media capability of contemporary BMCs coupled with virtual media boot support implemented in some ironic hardware types (e.g. *redfish-virtual-media*) fully solves problem (1). The rest of this spec is dedicated to pondering items (2) and (3).

[1] https://networks.nokia.com/products/flexi-zone

Gathering node configuration

Typical ironic node deployment workflow involves booting Ironic Python Agent (known as IPA or ramdisk) has to be booted to prepare the node. Once set up by IPA, the tenant (AKA instance) operating system is brought up.

In the context of OpenStack, existing cloud infrastructure is capable to manage tenant network configuration (e.g. Neutron), to deliver network configuration to the tenant (e.g. config-drive[2]) and even simplify the application of node configuration (e.g. cloud-init[3], os-net-config[4] or Ignition[11]). All these capabilities together largely solve problems (2) and (3) for the tenants.

However, our software infrastructure does not presently offer much in part of configuring ironic ramdisk networking without DHCP. This spec proposes:

- Allowing the operator to manually associate static node network configuration with ironic node object. The assumption is that the operator would use some other mechanism for IP address management.
- Enable ironic to leverage Neutron for building static network configuration for the node being deployed out of Neutron information.

Provisioning node configuration

This spec proposes burning the contents of *config-drive* containing Nova metadata into the ISO image the node has been booted from in provisioning and state. If no *config-drive* information is supplied to Ironic by the operator, ironic will create one.

To facilitate network configuration processing and application, this spec proposes reusing *network-data.json*[8] Nova metadata format, for Ironic-managed node network configuration.

Example *network-data.json* file:

```
{
    "links": [
        {
            "id": "interface0",
            "type": "phy",
            "ethernet_mac_address": "a0:36:9f:2c:e8:80",
            "mtu": 1500
        },
        {
            "id": "interface1",
            "type": "phy",
            "ethernet_mac_address": "a0:36:9f:2c:e8:81",
            "mtu": 1500
        }
    ],
    "networks": [
        {
```

(continues on next page)

[2] https://docs.openstack.org/nova/latest/user/metadata.html#config-drives
[3] https://cloudinit.readthedocs.io/en/latest/
[4] https://github.com/openstack/os-net-config
[11] https://github.com/coreos/ignition/blob/master/doc/configuration-v3_0.md
[8] https://specs.openstack.org/openstack/nova-specs/specs/liberty/implemented/metadata-service-network-info.html

(continued from previous page)

```
            "id": "provisioning IPv4",
            "type": "ipv4",
            "link": "interface0",
            "ip_address": "10.184.0.244",
            "netmask": "255.255.240.0",
            "routes": [
                {
                    "network": "10.0.0.0",
                    "netmask": "255.0.0.0",
                    "gateway": "11.0.0.1"
                },
                {
                    "network": "0.0.0.0",
                    "netmask": "0.0.0.0",
                    "gateway": "23.253.157.1"
                }
            ],
            "network_id": "da5bb487-5193-4a65-a3df-4a0055a8c0d7"
        },
        {
            "id": "provisioning IPv6",
            "type": "ipv6",
            "link": "interface1",
            "ip_address": "2001:cdba::3257:9652/24",
            "routes": [
                {
                    "network": "::",
                    "netmask": "::",
                    "gateway": "fd00::1"
                },
                {
                    "network": "::",
                    "netmask": "ffff:ffff:ffff::",
                    "gateway": "fd00::1:1"
                },
            ],
            "network_id": "da5bb487-5193-4a65-a3df-4a0055a8c0d8"
        }
    ],
    "services": [
        {
            "type": "dns",
            "address": "10.0.0.1"
        }
    ]
}
```

This spec anticipates associating the content of *network_data.json* document with ironic node object by introducing a new *network_data* field to the node object to contain *network_data.json* information for ironic ramdisk booting.

On the ramdisk side, this spec proposes using Glean[12] for consuming and applying network configuration to the operating system running ironic ramdisk. The main consideration here is that, unlike *cloud-init*, *Glean* is lean, and readily supports a subset of *cloud-init* features.

Alternative ramdisk implementations can choose other ways of bootstrapping OS networking based on *network_data.json* information.

To summarize - in the area of provisioning node network configuration this spec proposes:

- Reusing Nova metadata representation for provisioning network configuration via ramdisk image.
- Adding a new field to ironic node object: *network_data* to use for ramdisk bootstrapping.

  The contents of this field should be validated by ironic conductor API against *Glean* JSON schema and some ad-hoc checks the implementers deem reasonable.

  Having *Glean* schema effectively hardwired into ironic conductor API will not allow for an easy extension or addition of other network configuration formats.
- Creating a new *config-drive* to have it including *network-data.json* file.
- Writing the contents of *config-drive* image into the root of the ISO file system (along with ramdisk and kernel blobs), then making a bootable ISO image.
- Including *Glean* dependency to ramdisk image for managed OS bootstrapping.

However, Ironic rescue operation, at least in its current implementation, will only work if user and provisioning networks are the same network.

Thats because rescue ramdisk will try to renumber NICs of ramdisk by restarting DHCP client. Working around this limitation is out of scope of this spec.

Deployment workflow

To make it easier to grasp, lets reiterate on the entire Ironic deploy work flow focusing on how network configuration is built and used. We will consider two scenarios - stand-alone ironic and ironic within OpenStack cloud. In each scenario we will only consider deploy ramdisk and omit instance booting phases.

Stand-alone ironic

1. Operator supplies deploy ramdisk network configuration, in form of *network-data.json* contents, via *network_data* field of ironic node being deployed. The contents of *network_data.json* must comply to the JSON schema of *network_data.json* that *Glean* can consume.
2. Ironic conductor validates supplied *network-data.json* against *Glean* schema (that is supplied to ironic API program via configuration) and fails early if validation fails. *Glean* schema will not allow any properties of *network_data.json* that cant be applied to the OS by *Glean* even if these properties are valid as Nova metadata.
3. Ironic builds a new *config-drive* image and places *network-data.json* file, with contents taken from *network_data* field, at a conventional location within *config-drive* image.
4. To boot deploy ramdisk, ironic builds bootable ISO out of *deploy_kernel* and *deploy_ramdisk* also writing *config-drive* contents into the root of boot ISO image.

   Glean running inside ramdisk will try to mount virtual CD drive(s), in search for a filesystem labeled *config-2*, read *network_data.json* and apply network configuration to the OS.

[12] https://docs.openstack.org/infra/glean/

Ironic within OpenStack

1. Prior to booting ramdisk, unless operator-supplied network configuration already exists in *network_data* ironic node field, ironic gathers network configuration for each ironic port/portgroup, associated with the node being deployed, by talking with Neutron. Then ironic builds network configuration for ramdisk operating system in form of a *network-data.json* file.
2. Ironic builds a new *config-drive* image and places *network-data.json* file, as build at step (1), at a pre-defined location within *config-drive* image.
3. To boot deploy ramdisk, ironic builds bootable ISO out of *deploy_kernel* and *deploy_ramdisk* also writing *config-drive* contents into the root of boot ISO image.

   Glean running inside ramdisk will try to mount virtual CD drive(s), in search for a filesystem labeled *config-2*, read *network_data.json* and apply network configuration to the ramdisk operating system.

Alternatives

Alternatively to associating the entire and consistent *network_data.json* JSON document with ironic node object, *network_data.json* can be tied to ironic port object. However, experimental implementation revealed certain difficulties stemming from port-centric design, so consensus has been reached to bind *network_data.json* to ironic node object.

Alternatively to make ironic gathering and building *network-data.json*[Page 899, 8], ironic could just directly request Nova metadata service[10] to produce necessary file by instance ID. However, this will not work for non-deploy operations (such as node cleaning) because Nova is not involved.

Alternatively to relying on Nova metadata and *Glean* as its consumer in ramdisk, we could leverage *os-net-config*s feature of taking its compressed configuration from kernel parameters. On the flip side, the size of kernel cmdline is severely limited (256+ bytes). Also, *os-net-config* feels like a TripleO-specific tool in comparison with *cloud-init*, which, besides being a mainstream way of bootstrapping instances in the cloud, understands OpenStack network configuration metadata.

Alternatively to having operator supplying ramdisk network configuration as a *network_data.json* file, Ironic can also accept the entire *config-drive*. That would make it looking similar to instance booting (e.g. Ironic provision state API) and would allow for passing more files to ramdisk in the future.

Alternatively to wiring *Glean* schema validation into Ironic conductor, operator can be asked to validate their *network_data.json* data with some external tool prior to submitting it to Ironic. This would relax ironic conductor dependency on *Glean* input requirements changes and ease *network_data.json* reuse for bootstrapping both ramdisk and instance.

Data model impact

Add a new, user manageable, field *network_data* to ironic node object conveying ramdisk network configuration information to ironic. If set, the contents of this new field should be a well-formed *network-data.json* document describing network configuration of the node running ramdisk.

[10] https://docs.openstack.org/nova/latest/user/metadata-service.html

State Machine Impact

None.

REST API impact

- Update `GET /v1/nodes/detail`, `GET /v1/nodes/{node_id}`, `PATCH /v1/nodes/{node_id}` to add new request/response fields
- `network_data` JSON object conveying network configuration.

Client (CLI) impact

ironic CLI

None

openstack baremetal CLI

- Update `openstack baremetal node create` and `openstack baremetal node set` commands to accept new argument `--network-config <JSON>` with help text describing JSON structure of the network configuration.
- Extend the output of the `openstack baremetal node show` command with `network_data` column.

RPC API impact

None

Driver API impact

- Extend ironic base NetworkInterface with *get_node_network_config* API call providing network configuration for the node being managed. Ironic will burn the output of this API call to the boot image served over virtual media.
- Implement *get_node_network_config* network interface call for non-Neutron networks providing *network_data.json* from *network_data* field of ironic object (if present). The operator could then implement their own IPAM (e.g. for stand-alone ironic use-case).
- Implement *get_node_network_config* network interface call for Neutron networks generating *network_data.json* based on Neutron port and subnet information[9].
- Make virtual media boot interfaces in ironic generating config-drive with *network_data.json* in it if *network_data.json* is not already passed to ironic with the config-drive.
- Make virtual media boot interfaces in ironic writing config-drive contents into root of bootable ISO image it generates on every node boot. Filesystem on this bootable ISO should be labeled *config-2* if it contains config-drive files.

[9] https://github.com/openstack/nova/blob/master/nova/virt/netutils.py#L60

Nova driver impact

None

Ramdisk impact

- Diskimage-builder tool should install *Glean* into ramdisk and invoke on boot.

  On top of that, the *dhcp-all-interfaces* DIB element might not be used anymore because *Glean* will run DHCP on all not explicitly configured (via *config-drive*) interfaces[13].

Security impact

None.

Other end user impact

None.

Scalability impact

None.

Performance Impact

None.

Other deployer impact

None.

Developer impact

None.

Implementation

Assignee(s)

Primary assignee:
etingof (etingof@gmail.com)

Other contributors:
None.

Work Items

- Document *Glean* requirements, with regards to *network_data.json*, in form of machine-readable JSON schema.
- Update ironic node model to include optional, user-specified *network_data* fields carrying ramdisk network configuration in form of *network_data.json* JSON document.

[13] https://opendev.org/opendev/glean/src/branch/master/glean/cmd.py#L323

- Update REST API endpoints to support new *network_data* field
- Support new *network_data* fields in baremetal CLI (*openstack baremetal node*)
- Extend ironic base NetworkInterface with the *get_node_network_config* API call providing network configuration for the node being managed.
- Implement *get_node_network_config* network interface call for non-Neutron networks providing *network_data.json* from *network_data* field of ironic node object (if present).
- Implement *get_node_network_config* network interface call for Neutron networks generating *network_data.json* based on Neutron port and subnet information[Page 903, 9] .
- Make virtual media boot interfaces in ironic generating *config-drive* with *network_data.json* on it.
- Make virtual media boot interfaces in ironic writing config-drive files into the root file system of the bootable ISO image it generates on every node boot. The file system should be labeled as *config-2* for *Glean* to find and use it.
- Update ramdisk bootstrapping code to invoke *Glean* on system boot making use of *network-data.json* file if present on the *config-drive*.
- Update diskimage-builder tool to control the inclusion and the options of the new static network configuration processing features.
- Create documentation on DHCP-less boot setup and work flow.

Dependencies

Ramdisk will start depending on *Glean* for processing *network-data.json* document.

Testing

Tempest test of the ironic node deployment using operator-supplied and Neutron-originated *network_data.json*.

Upgrades and Backwards Compatibility

None.

Documentation Impact

Use of L3 based deployment should be documented as part of this item.

References

5.15.100 Active Node Creation

https://bugs.launchpad.net/ironic/+bug/1526315

This spec is intended to allow a slightly more permissive interaction with the ironic API to allow an operator to migrate a hardware fleet to be managed by ironic.

Problem description

At present the ironic API explicitly sets the new state for nodes to the beginning step in the ironic workflow.

As part of hardware fleet lifecycle management, an operator expects to be able to migrate inventory and control systems for their hardware fleet utilizing their existing inventory data and allocation records.

Ultimately this means that an imported host MAY already be allocated and unavailable for immediate allocation.

For an operator of multiple distinct OpenStack infrastructures, it is reasonable to permit an operator to migrate running baremetal hosts from one system to another system that are ultimately components in a larger infrastructure, while not immediately reprovisioning hardware.

Proposed change

Allow an API client to transition a node directly from the `MANAGEABLE` state to the `ACTIVE` state, bypassing actual deployment of the node.

- Creation of a new API provision_state verb of `adopt` that invokes the state transition of `ADOPTING`.
- Creation of a new machine state transition of `ADOPTING` which is valid only in the `MANAGEABLE` state and allows an operator to directly move a node to `active` state. This transition would be dependent upon a successful interface validation. Failure of this transition shall move nodes to an `ADOPTFAIL` which will allow users to be able to identify nodes that failed adoption.
- Creation of a new machine state of `ADOPTFAIL` which a machine is set to upon the `ADOPTING` transition failing. This state will allow a user to re-attempt `ADOPTING` via `adopt`, or attempt to transition the node to the `MANAGEABLE` state via `manage`. Additionally, the `ADOPTFAIL` state will be listed in the list of states that permit node deletion from the ironic database.
- API client update to provide CLI interface to invoke this feature.
- Creation of explicit documentation covering:

```
- Use cases of the feature while explicitly predicating that proper
  operation requires node validation to succeed.
- Explicitly detail that it is the operator's responsibility to
  define the node with all relevant appropriate configuration else
  the node could fail node state provision operations of ``rebuild``
  and ``delete``. Which would result in manual intervention being
  necessary.
- Explain the basic mechanics of the use of the adoption feature
  to users in order to help convey the importance of the correct
  information being populated.
```

Alternatives

The logical alternative is to remove restrictions in what an API client posts to allow the caller to explicitly state or invoke a node to be created in `ACTIVE` state. As the community desires full functionality of the node to exist upon being imported along with driver interface validation, the implementation appears to lend itself to be implemented as a state transition instead of pure API logic.

Alternatively, we could craft operator documentation to help assist operators in directly loading nodes into the ironic database, coupled with the caveats of doing so, and require that documentation is updated in lock-step with any database schema changes.

Data model impact

None

State Machine Impact

Implementation of a new state transition from `MANAGEABLE` state to `ACTIVE` state utilizing an intermediate state of `ADOPTING` which takes the following actions.

1. Triggering the conductor node take_over logic.
2. Upon completion the node state is set to `ACTIVE`.

Should a failure of take_over logic occur in the `ADOPTING` state, the node will be returned to `ADOPTFAIL` state from which a user will be able to retry the adoption operation or delete the node.

Addition of `ADOPTFAIL` into the `DELETE_ALLOWED_STATES` list.

REST API impact

Addition of a new state verb of `adopt` that triggers a transition to `ADOPTING` state. This verb will be unavailable for clients invoking an insufficient API version.

The API micro-version will need to be incremented as a result of this change.

Client (CLI) impact

Update of the ironicclient CLI to detail that `adopt` is a possible provision state option.

Update of the ironicclient micro-version.

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

N/A

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

Minimal API impact will exist for a user of this feature as the creation of nodes in `ACTIVE` state will require multiple calls with the API by any user attempting to leverage this feature.

Users performing bulk loads of hosts may find the multiple API calls somewhat problematic from the standpoint of multiple API calls to create, validate, and adopt a node, on top of API calls to poll the current state of the node before proceeding to the next step. Bulk loaders should also be congnizant of their configurations as they potentially could result in the conductors consuming disk space and network bandwidth if items need to be staged on the conductor to support the nodes normal operation.

Other deployer impact

Allows for an easier adoption by managers of pre-existing hardware fleets.

There is the potential that a operator could define a hardware fleet with bare minimal configuration to initially add the node to ironic. The result of which means that an operator could conceivably and inadvertently act upon a node when insufficient information is defined. This risk will be documented as part of the resulting documentation in order to help highlight the risk and help provide guidance on preventing such a possibility should a user be attempting to adopt an inventory that is already cloudy.

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
juliaashleykreger

Other contributors:
None

Work Items

- Conductor State Machine Changes
- API microversion and update and appropriate logic
- CLI node-set-provision-state option addition
- Documentation updates

Dependencies

None

Testing

Addition of unit tests as well as tempest tests to explicitly test the interface.

Upgrades and Backwards Compatibility

This feature will not be usable by older API clients via the API micro-version interface.

Documentation Impact

Documentation will need to be updated to represent this new feature.

References

None

5.15.101 Add a field to accept the default verify_ca path

https://bugs.launchpad.net/ironic/+bug/2040236

This spec adds an option for setting [driver]verify_ca for each driver. This value would then be used by Ironic for certificate validation instead of the default system CA bundle for that driver. Each driver may use a different CA certificate for this purpose if desired.

Problem description

Currently, Ironic utilizes a system-wide CA certificate bundle to validate HTTPS connections. However, there are scenarios where this default behavior may not align with the specific requirements or policies of an organization. Particularly, when a user specifies that certificate validation is required, administrators may prefer not to rely on the systems default certificate path for authentication with BMCs. Instead, they might need to utilize custom CA certificates, which are tailored to their organizational security policies or specific use cases.

At present, if the operator desires to configure a custom CA certificate for communications between Ironic and the BMCs of managed bare-metal servers, they have the ability to configure the <driver>_verify_ca option on the node or nodes in question. This option can be configured to be either true, false, or set to a specific certificate path. When <driver>_verify-ca is set to true, Ironic defaults to using the systems CA bundle for certificate validation. Furthermore, in cases where the operator desires to specify a CA certificate outside system CA bundle, this approach has a serious shortcoming: it requires the operator to know the exact location of the pre-existing CA certificate in the filesystem. While this configuration option is set on per-node level through Ironic APIs, there is no way for the operator to either upload the CA certificate through those APIs or to determine the filesystem path of such certificate. This is not a desirable situation. This option is also unsuitable for the use cases where the operator is only able to interact with Ironic through writing configuration file and making API calls (which is the case in metal3).

To overcome this limitation, we propose a new configuration option scoped on a driver level (as opposed to the node level): [driver]verify_ca. This way the required CA certificates can be deployed to known locations at the deploy time and Ironic configuration for each driver can be written referencing those paths.

Proposed change

1. Adding a new option verify_ca for each driver

- Adding a verify_ca option to each drivers conf to accept the specified value. Making this option configurable on a per-driver level is consistent with existing configuration options (e.g. kernel_append_params). It is also very clear what purpose this specific CA certificate serves.

2. Retrieving the path before node verification

- Before performing the node verification, retrieve the certificate path, and pass it to verify_ca for validation. This implementation vary based on different vendors.

Alternatives

- Administrators may need to log in to the system where ironic is located and manually add the desired certificate. Note: this requires modifications of the trusted CA certificate collection on the machine running Ironic, which may or may not be desired; there may be use cases where an operator wishes to trust a certain CA for connections to BMC but not other encrypted communications involving the server running Ironic.
- Instead of [driver]verify_ca, a global configuration option similar to [conductor]verify_ca could be considered, however this could lead to confusion about the impact of setting it (does it apply to all conductor connections? BMC connections? else?) and would be inconsistent with other, similar options (e.g. kernel_append_params).
- Using the existing <driver>_verify_ca setting is another alternative, however it requires prior existence of the desired CA certificate in the filesystem where Ironic is running as well as operators knowledge of the exact path, which should not be relied upon and as such is not desired.

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

None

Security impact

The introduction of the verify_ca configuration option in Ironics driver could have several security implications:

1. Handling of Sensitive Data:

- This change involves the path to CA certificates, which are critical for secure communications. While the option itself doesnt handle sensitive data like tokens or keys, it directs Ironic to the location of sensitive cryptographic material. Proper management and permissions of this path are essential to prevent unauthorized access.

2. Accessibility of Managed Hardware:

- By allowing a custom CA path, this change could affect the security of the hardware managed by Ironic. If an incorrect or untrusted CA path is specified, it could potentially compromise the integrity of the SSL/TLS connections with the BMCs, impacting the secure management of the hardware.

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

- At deploy time, one or more custom CA certificates may be installed on the machine running Ironic under a known path.
- Ironic configuration created at the deploy time will assign the custom CA certificate to the driver(s) that are expected to be using it.
- Each driver can use a different CA certificate, or the same CA certificate may be used by multiple drivers if desired.

- The default None value for verify_ca ensures backward compatibility, using the systems default CA bundle unless overridden. This approach maintains operational stability for existing deployments while offering flexibility for custom configurations.
- Ironic will be enhanced to dynamically retrieve the value of verify_ca for each hardware driver and pass it to the verify_ca function. This mechanism ensures that SSL/TLS communications with BMCs across different hardware types can leverage the specified custom CA certificates.
- The feature requires explicit enablement by deployers. This change will not automatically activate.

Developer impact

- Developers will need to ensure that their drivers correctly interpret and utilize the specified CA path.

Implementation

Assignee(s)

Primary assignee:
Zhou Hao <zhouhao@fujitsu.com>

Other contributors:
Zou Yu <zouy.fnst@fujitsu.com> Feng GuangWen <fenggw-fnst@fujitsu.com>

Work Items

- Implement option verify_ca for each vendor.
- Update documentation.

Dependencies

None

Testing

- Test the driver to verify that the set path is used when performing certificate validation.

Upgrades and Backwards Compatibility

None

Documentation Impact

- The documentation should be updated for each hardware vendor as features are implemented.

References

None

5.15.102 Add InfiniBand Support

https://bugs.launchpad.net/ironic/+bug/1532534

Today, Ironic supports Ethernet interfaces for Hardware inspection and PXE boot. Ironic should have the ability to inspect and PXE boot over InfiniBand network as well.

Problem description

- Hardware inspection for InfiniBand - A InfiniBand GUID is similar in concept to a MAC address because it consists of a 24-bit manufacturers prefix and a 40-bit device identifier (64 bits total).
- PXE Boot over InfiniBand - To allow DHCP over IPoIB interface the DHCP client must send the client-id with a unique identifying value for the client. The value is consist of the vendor prefix 12 bytes and GUID 8 bytes. Today ironic doesnt send update the neutron port with client-id option.

Proposed change

- Hardware inspection for InfiniBand - In order to use InfiniBand with PXE you have to flash the NIC with a vendor specific firmware. The vendor firmware defined the conversion of GUID to InfiniBand MAC (48 bits). To resource complicity of the change, the ironic address will contain the InfiniBand MAC.
- PXE/iPXE Boot over InfiniBand changes: To allow DHCP over InfiniBand we need the following:
  1. The dhcp-server must use the BROADCAST flag in the dhcp-server. This already support in neutron-dhcp-agent by config file.
  2. Updating the ironic port extra attribute to contains the InfiniBand port client-id extra e.g:

     ```
     {
         'client-id':
         'ff:00:00:00:00:00:02:00:00:02:c9:00:00:02:c9:03:00:00:10:39'
     }
     ```

     The client-id update can be done manually or with IPA and ironic-inspector.
  3. The neutron port that represent the ironic port should be updated with client-id option in the extra_dhcp_opts attribute. The client-id consists of a vendor prefix and the port GUID. The client id for Mellanox ConnectX Family Devices is consists of a prefix (ff:00:00:00:00:00:02:00:00:02:c9:00) and 8 byte port GUID. The prefix in the client-id is vendor specific.
  4. The PXE MAC file name consists of the <Hardware Type>-<MAC>. For InfiniBand the hardware Type is 20 and the mac is the InfiniBand truncate GUID.
  5. The iPXE MAC file name consists of the <MAC>. For InfiniBand the MAC is the InfiniBand truncate GUID.

Other projects changes:

- ironic-python-agent changes:
  1. Update the ironic agent to calculate the InfiniBand truncate GUID and the Client ID.
  2. Update coreos and tinyipa with ib_ipoib driver.
- ironic-inspector changes:
  1. Update the ironic-inspector to update port.extra with client-id
- diskimage-builder changes:
  1. Update the mellanox element to load ib_ipoib driver.

Alternatives

- Extend the ironic port to support GUID which is 8 bytes and calculate the client-id in the ironic code from the GUID. This will require updating the ironic model and API. This will require updating the nova ironic driver to truncate the GUID to MAC.

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

ironic CLI

None

openstack baremetal CLI

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

N/A

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

When using IPA, the deployer needs IPA that provides the InfiniBand MAC and client-id.

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
moshele

Other contributors:
None

Work Items

- Add Client-ID option to the neutron port to allow DHCP.
- Update the generation of the iPXE/PXE file.
- Update documentation.

Dependencies

None

Testing

- Adding unit tests.
- Adding Third-party CI which will test Mellanox hardware.

Upgrades and Backwards Compatibility

None

Documentation Impact

- We will update the ironic documentation on how to allow pxe boot from IPoIB.

References

- http://www.syslinux.org/wiki/index.php/PXELINUX
- https://tools.ietf.org/html/rfc4392
- http://www.mellanox.com/related-docs/prod_software/Mellanox_FlexBoot_User_Manual_v2.3.pdf

5.15.103 Add more capabilities to ironic inspection

https://bugs.launchpad.net/ironic/+bug/1599425

This spec adds a few more capabilities to ironic drivers. The capabilities can be implemented in-band or out-of-band as per driver maintainers technical decision.

Problem description

The operator may be interested to schedule and select the hardware based on many other hardware properties like cpu_vt, trusted_boot, etc.

Proposed change

Following properties will be discovered:

```
capability name: trusted_boot
possible values: true|false
explanation    : The hardware supports trusted boot or not.

capability name: iscsi_boot
possible values: true|false
explanation    : The hardware supports iscsi boot or not.

capability name : boot_mode_uefi
possible values : true|false
explanation     : The hardware supports uefi boot or not.
                  It is not the current boot mode. This
                  may not be discovered through inband
                  inspection.

capability name : boot_mode_bios
possible values : true|false
explanation     : The hardware supports bios boot or not.
                  It is not the current boot mode.

capability name : sriov_enabled
possible values : true|false

capability_name : has_ssd
possible values : true|false
```

(continues on next page)

(continued from previous page)

```
capability_name : has_rotational
possible_values : true|false

capability name : rotational_drive_<rpm_value>_rpm
possible values : true|false
explanation     : The capabilities will turn out to be
                  rotational_drive_4800_rpm,
                  rotational_drive_5400_rpm,
                  rotational_drive_7200_rpm,
                  rotational_drive_10000_rpm, and
                  rotational_drive_15000_rpm. These
                  looks to be the only and standard values
                  for rotational drive rpms.

capability name : logical_raid_level_<num>
possible values : true|false
explanation     : The capabilities ``logical_raid_level_<num>``
                  will have "num" as the dynamic number and
                  will have names as ``logical_raid_level_1``,
                  ``logical_raid_level_2``, and so on. There can
                  be multiple RAIDs configured on a hardware. This
                  gives flexibility to the operator to choose a
                  hardware which has the desired raid configured.
                  So if RAID level 1 is configured, the
                  variable becomes ``logical_raid_level_1`` set
                  to ``true``. if RAID level 5 is configured,
                  the variable becomes ``logical_raid_level_5``
                  set to ``true``. These capabilities would be
                  used only for scheduling, and ironic is
                  not supposed to create RAID level as per these
                  capabilities.
                  These capabilities should be added via RAID
                  interfaces also. These are added via inspection
                  as there can be baremetals added to ironic which
                  have RAID pre-configured.

capability name : cpu_vt
possible values : true|false

capability name : hardware_supports_raid
possible values : true|false

capability name : boot_mode
possible values : bios, uefi
explanation     : This represents the deploy boot mode of
                  the system.

capability name : has_nvme_ssd
```

(continues on next page)

(continued from previous page)

```
possible values : true|false

capability name : persistent_memory
possible values : true|false

capability name : nvdimm_n
possible values : true|false

capability name : logical_nvdimm_n
possible values : true|false
```

The other drivers/vendors may require the below list of capabilities. These are already implemented by iLO drivers:

```
capability name : <driver>_firmware_version
possible values : varies from hardware to hardware
explanation     : Here driver means ilo or irmc or any other
                  vendor. Hence the capability becomes
                  ilo_firmware_version or irmc_firmware_version.

capability name : server_model
possible values : varies from hardware to hardware.

capability name : secure_boot
possible values : true|false

capability name : pci_gpu_devices
possible values : count of total GPU devices.

capability name : nic_capacity
possible values : Maximum NIC capacity value with unit.

capability name : rom_firmware_version
possible values : vary from hardware to hardware.
```

Few which are already implemented by ironic-inspector:

```
capability name : cpu_aes
possible values : true|false

capability name : cpu_txt

capability name : cpu_hugepages

capability name : cpu_hugepages_1g
```

These may not be part of capabilities specifically, but required to be inspected. The ironic-inspector already inspects these properties:

```
property name   : switch_id
```

(continues on next page)

(continued from previous page)

```
explanation     : Identifies a switch and can be a MAC address
                  or an OpenFlow-based ``datapath_id``

property_name   : port_id
explanation     : Port ID on the switch, for example, Gig0/1

property_name   : switch_info
explanation     : Used to distinguish different switch models
                  or other vendor-specific identifier.
```

`has_ssd` and `has_rotational` are two different properties as the hardware can have both kind of drives attached.

The capabilities `boot_mode_*` are added as a hardware could be supporting both bios and uefi and the current capability `boot_mode` can accept only one value. The drivers would need to adjust the deploy behaviour when the new capabilities `boot_mode_bios` or `boot_mode_uefi` are given in nova flavor. The changes required in drivers for `boot_mode_*` capabilities is out of scope of this spec.

Alternatives

Operator may need to manually configure the node with above properties for nova scheduler to be able to select the desired node for deploy.

Data model impact

None.

State Machine Impact

None.

REST API impact

None.

Client (CLI) impact

None.

RPC API impact

None.

Driver API impact

None.

Nova driver impact

None.

Ramdisk impact

None.

Security impact

None.

Other end user impact

None.

Scalability impact

None.

Performance Impact

None.

Other deployer impact

None.

Developer impact

None.

Implementation

Assignee(s)

Primary assignee:
Nisha Agarwal <agarwalnisha1980>

Work Items

- To add the above capabilities to the inspection.

Dependencies

None.

Testing

- Test the drivers to return above properties after inspection is done.

Upgrades and Backwards Compatibility

None.

Documentation Impact

Following will be documented:

- The new properties which would be added as part of this spec.
- The nova flavor samples how these properties can be used in creation of required nova flavors.

References

None.

5.15.104 Add pluggable metrics backend for Ironic and IPA

https://bugs.launchpad.net/ironic/+bug/1526219

This proposes the addition of metric data reporting features to Ironic, and Ironic Python Agent (IPA). Initially, this will include a statsd reference implementation, but will be sufficiently generic to permit the creation of alternative backends.

Problem description

Software metrics are extremely useful to operators for recognizing and diagnosing problems in running software, and can be used to monitor the real time and historical performance of Ironic and IPA in a production environment.

Metrics can be used to determine how quickly (or slowly) parts of the system are running, how often errors (such as API error responses or BMC failures) occur, or the performance impact of a given change.

Currently, neither Ironic nor IPA report any application metrics.

Proposed change

- Design a shared pluggable metric reporting system.
- Implement a generic MetricsLogger which includes:
  - Gauges (generic numerical data).
  - Counters (increment/decrement a counter).
  - Timers (time something).
  - Decorators, and context managers for same.
- Implement a StatsdMetricsLogger as the reference backend [1].
- Instrument Ironic to report metric data including:
  - Counting and timing of API requests. This may be accomplished by hooking into Pecan.
  - Counting and timing of RPCs.
  - Counting and timing of most worker functions in ConductorManager.
  - Counting and timing of important driver functions.

  - Count and time node state changes. By inspecting provision_updated_at during a state change, the time the node spent in that state can be calculated.

- Instrument IPA to report metric data including, but not limited to:
  - Image download/write counts and times.
  - Deploy/cleaning counts and times.

Example code follows (based on Python logging module naming conventions):

```
METRICS = metrics.getLogger(__name__)

class Foo(object):
  def func1(self):
    # Emit gauge metric with value 1
    METRICS.send_gauge("one.fish", 1)

    # Increment counter metric by two
    METRICS.send_counter("two.fish", 2)

    # Decrement counter metric by one
    METRICS.send_counter("red.fish", -1)

    # Randomly sample the data (emit metric 10% of the time)
    METRICS.send_counter("blue.fish", 42, sample_rate=0.1)

    # Emit a timer metric with value of 125 (milliseconds)
    METRICS.send_timer("black.fish", 125)

    # Randomly sample the data (emit metric 1% of the time)
    METRICS.send_timer("blue.fish", 125, sample_rate=0.01)

  @METRICS.counter("func2.count")
  @METRICS.timer("func2.time", sample_rate=0.1)
  def func2(self):
    pass

  # Context managers for counting and timing code blocks
  def func3(self):

    with METRICS.counter("func3.thing_one.count", sample_rate=0.25):
      thing_one()

    with METRICS.timer("func3.thing_two.time"):
      thing_two()
```

Metric names follow this convention (optional parts indicated by []):

`[global_prefix.][host_name.]prefix.metric_name`

If *metrics-agent-prepend-host-reverse* is set, then `host.example.com` becomes `com.example.host` to assist with hierarchical data representation.

For example, using the Statsd backend, and relevant config options, `METRICS.send_timer("blue.`

fish", 125, sample_rate=0.25) is emitted to statsd as globalprefix.com.example.host.moduleprefix.blue.fish:1|ms@0.25.

Alternatives

Alternatively, we could implement a Ceilometer backend. Although Ironic already reports some measurements (such as IPMI sensor data) to Ceilometer, the metrics that are proposed in this spec do not fit with the Ceilometer project mission, which is to collect measurements of the utilization of the physical and virtual resources comprising deployed clouds [2]

Instead, this spec proposes that we instrument parts of the Ironic/IPA codebase itself to report metrics and statistics about how/when the code is run, and the performance of the code thereof. This data is not directly related to physical and virtual resources comprising deployed clouds. Therefore, we do not propose the addition of a Ceilometer backend, nor do we propose that the existing Ceilometer measurements be converted to this system, as they represent fundamentally different types of data.

Data model impact

None

State Machine Impact

None.

REST API impact

To support agent drivers, a config field will be added to the response for the /drivers/<drivername>/vendor_passthru/lookup endpoint in the Ironic API.

This field will contain the agent-related config options that an agent can use to configure itself to report metric data. For example: statsd host and statsd port.

Client (CLI) impact

None.

RPC API impact

None.

Driver API impact

None.

Nova driver impact

None.

Ramdisk impact

N/A

Security impact

The statsd daemon [3] has no authentication, and consequently anyone who is able to send UDP datagrams to the daemon can send arbitrary metric data. However, the statsd daemon is typically configured to listen only on a local interface, which partially mitigates security concerns.

Other end user impact

None.

Scalability impact

Deployers must ensure that their statsd infrastructure is scaled correctly relative to the size of their deployment. However, even if the statsd daemon is overloaded, Ironic will not be negatively affected (statsd UDP datagrams are non-blocking, and will simply not be processed).

Performance Impact

By default, metrics reporting will be disabled, reducing, but not totally eliminating, the performance impact for users who do not wish to collect metrics. At the very least, a conditional must be checked at each place where a metric could be reported. Furthermore, depending on exactly how and where the conditional checking occurs, arguments may be evaluated even if the metric data arent actually sent.

Reporting metrics via statsd affects performance minimally. The overhead of sending a single piece of metric data is very smallin particular, statsd metrics are sent via UDP (non-blocking) to a daemon [2] that aggregates the metrics before forwarding them to one of its supported backends. Should this backend become unresponsive or overloaded, then metric data will be lost, but without other performance effects.

After the metric data are aggregated by a local statsd daemon, they are periodically flushed to one of statsds configured backends, usually Graphite [4].

Other deployer impact

There are two different sets of configuration options to be added:

These options will be set in the ironic-lib metrics library, and will be used by any ironic service implementing metrics:

```
[metrics]

# Backend options are "statsd" and "noop"
backend="noop"
statsd_host="localhost"
statsd_port=8125

# See proposed changes section for detailed description of how these are used
prepend_host=false
prepend_host_reverse=false
global_prefix=""
```

Additionally, the following options will be added to the ironic-conductor and used to configure the ironic-python-agent for metrics on lookup:

```
# Backend options are "statsd" and "noop"
agent_backend="noop"
agent_statsd_host="localhost"
agent_statsd_port=8125

# See proposed changes section for detailed description of how these are used
agent_prepend_host=false
agent_prepend_host_reverse=false
agent_prepend_uuid=false
agent_global_prefix=""
```

If the statsd metrics backend is enabled, then deployers must install and configure statsd, as well as any other metrics software that they wish to use (such as Graphite [3]). Additionally, if deployers wish to emit metrics from ironic-python-agent as well, the statsd backend must be accessible from networks that agents run on.

Developer impact

None.

Implementation

Assignee(s)

Primary assignee:
alineb

Other contributors:
JayF

Work Items

- Design/implement metric reporting into ironic-lib.
- Implement statsd backend.
- Instrument Ironic code to report metrics.
- Instrument IPA code to report metrics.

Dependencies

None.

Testing

Additional care may be required to test the statsd network code.

Upgrades and Backwards Compatibility

None.

Documentation Impact

Appropriate documentation must be written.

References

For more on why metrics are useful to operators, and why the statsd project began: https://codeascraft.com/2011/02/15/measure-anything-measure-everything/

[1] https://github.com/etsy/statsd/blob/master/docs/metric_types.md

[2] https://wiki.openstack.org/wiki/Ceilometer

[3] https://github.com/etsy/statsd/

[4] https://graphite.readthedocs.org/en/latest/faq.html

5.15.105 Promote agent vendor passthru to core API

https://bugs.launchpad.net/ironic/+bug/1570841

This spec suggests making the current agent vendor passthru API (lookup and heartbeat) first class API endpoints and deprecate the agent vendor passthru interface.

Problem description

The vendor passthru was designed for vendors to place their specific features before they get wider adoption in Ironic and get promoted to the core API.

However, when IPA is used (which is the only ramdisk available in-tree), these two API endpoints play the critical role in both deployment and cleaning processes. Thus every IPA-based driver must mix agent vendor passthru into its vendor passthru.

There was a bug in the drac driver when it was not done, and the driver did not work with IPA.

This proposal also tries to reduce the amount of data sent to and from unauthenticated endpoints. The current vendor passthru API accepts the whole inventory and returns the whole node record, including IPMI credentials.

Proposed change

- Create new API endpoints for lookup and heartbeat - see *REST API impact* for details.
- Extend the deploy interface with a heartbeat method - see *Driver API impact* for details.

Alternatives

- Continue doing what we do now.
- Make the lookup call driver-dependent (as the passthru used to be).

  This looks like unnecessary complication (e.g. we have to pass a driver to IPA from the conductor nowadays).
- We could use this change to move the unauthenticated endpoints away from the main API completely. This could be done by introducing a new API service, say `ironic-agent-api`, serving only these two endpoints. Then we will recommend operators to make this service only listen on the provisioning network, but not on the network accessible to users.

Arguably the same result can be achieved by configuring a WSGI container (Apache mod_wsgi or similar), so it might not be worth complication.

Data model impact

None

State Machine Impact

None

REST API impact

Two new endpoints are added. Both endpoints are **NOT** authenticated.

- `GET /v1/lookup?addresses=MAC1,MAC2&node_uuid=UUID`

  Look up node details for further use in the ramdisk.

  No body; at least one of the following URL parameters must be present:

  - `addresses` comma-separated list of hardware addresses (e.g. MAC) from the node for lookup;
  - `node_uuid` node UUID, if known (e.g. by inspection).

  If `node_uuid` is present, `addresses` are ignored.

  By default only return a node if its in one of transient states: `deploying`, `deploy wait`, `cleaning`, `clean wait`, `inspecting`, `inspect wait`. Deployers who need lookup to always work will be able to set a new option `[api]restrict_lookup` to `False`.

  Note

  In theory, we dont need `-ing` states here either. But when we reboot during cleaning, we dont currently reset the state to `clean wait`. The other `-ing` states are supported in case 3rd party drivers have similar restrictions.

  Response: HTTP 200 with JSON body containing keys:

  - `config` dictionary for passing configuration options from conductor to the ramdisk. For the IPA ramdisk only one is currently used:
    - `heartbeat_timeout` timeout (in seconds) between heart beats from the ramdisk, expected by Ironic.
  - `node` partial node representation as a JSON object, with the following fields sent:
    - `properties` for root device hints,
    - `instance_info` for disk sizing details,
    - `uuid` node UUID,
    - `driver_internal_info` for passing other runtime information.

    More fields can be exposed with time with an appropriate API version bump.

  Error codes:

  - 400 - bad request,
  - 404 - a node was not found.

- `POST /v1/heartbeat/<UUID>`

  Record a heartbeat message from the ramdisk.

  Body is a JSON with fields:

  - `callback_url` - the IPA URL to call back. Note that for potential non-IPA-based drivers it might have a different meaning (e.g. if we agree on the ansible driver, this can be an SSH URL for it).

  Response: HTTP 202 with no body.

  Error codes:

  - 400 - bad request,
  - 404 - a node was not found,
  - 409 - node is locked (should be retried by the ramdisk).

A new API version will be introduced to cover both endpoints.

Client (CLI) impact

Both endpoints will be exposed in the Python API for the ironic client as:

```
ironic.node.lookup(addresses, node_uuid=None)
ironic.node.heartbeat(node_uuid, callback_url)
```

However, as they are not intended for end users, they will not be exposed in both CLI.

ironic CLI

None

openstack baremetal CLI

None

RPC API impact

A new RPC call is created to connect the heart beat API endpoint and the new deploy driver method: `heartbeat` (async).

Driver API impact

A new method is added to the deploy driver interface:

```
def heartbeat(self, task, callback_url):
    """Record a heart beat for the node.

    :param task: a task manager task
    :param callback_url: a URL to use to call to the ramdisk
```

(continues on next page)

(continued from previous page)

```
    :return: None
    """
    LOG.warning('Got heartbeat message from node %(node)s, but the driver '
                '%(driver)s does not support heartbeating',
                {'node': task.node.uuid, 'driver': task.node.driver})
```

The heartbeat method from `BaseAgentVendor` will be refactored to a separate mix-in class for reusing in both `AgentDeploy` and `BaseAgentVendor`.

The new method will not be abstract to allow drivers that use a different approach (e.g. which do not have a ramdisk at all). The default implementation will do nothing to account for deploy drivers which do not need heart beats.

The new method will receive a shared node lock. It is up to the implementation to upgrade the lock to exclusive, if required.

Nova driver impact

None

Ramdisk impact

None

Security impact

- This change will expose unauthenticated API to lookup a node by its MAC addresses. It does not have any impact on most deployments, as both in-tree deploy methods (iscsi and http) already expose such API.
- After the complete switch to the new API endpoints is finished, it will no longer be possible to fetch the whole node knowing its MAC address without authentication. Only limited fields will be available. Notably, the power credentials are not sent in the new API endpoints.
- We should clearly note that any deploy implementation should treat the incoming data in the new `heartbeat` call with care. Particularly, no sensitive information should be ever sent to the endpoint designated by the `callback_url` parameter.

Other end user impact

None

Scalability impact

None

Performance Impact

- Unlike the old lookup passthru, the new lookup endpoint will not use RPC, lowering load on the message queue and the conductor.

Other deployer impact

- An update of the IPA image will be recommended to make it use the new API.
- New option `restrict_lookup` in the `api` section (boolean, defaults to `True`) - whether to restrict the new lookup API to only certain states in which lookup is expected.

Developer impact

3rd party driver developers should stop using the `BaseAgentVendor` class in their drivers and just use the `AgentDeploy` class.

3rd party drivers should document whether they require the `restrict_lookup` option to be `False` for correct functioning.

Implementation

Assignee(s)

- Dmitry Tantsur (lp: divius, irc: dtanstur)
- Jim Rollenhagen (irc: jroll)

Work Items

- Create new deploy interface methods
- Implement them in the AgentDeploy
- Create new RPC calls and API endpoints
- Switch IPA to use the new endpoints, and fall back to old ones on failure

Dependencies

None

Testing

Testing will be conducted as part of the current gate tests.

Upgrades and Backwards Compatibility

Full backward compatibility will be guaranteed independent of upgrade order between IPA and ironic itself.

The `BaseAgentVendor` class will be deprecated, but stay for some time, following the usual deprecation policy. Old IPA images will be able to run by using the old passthru API.

The new IPA image will try to hit the new endpoints first, and will fall back to the old ones on getting HTTP 406 Not Acceptable (meaning, the API version is not supported).

Documentation Impact

- Document how to implement new deploy drivers with the new heartbeat method.
- Document the potential security issues with both endpoints.

References

5.15.106 The direct deploy interface provisioning with HTTP server

https://storyboard.openstack.org/#!/story/1598852

This spec proposes a mechanism to provision baremetal nodes by hosting custom HTTP service as an image source provider to the `direct` deploy interface, when the Image service is utilized.

Problem description

Currently the `direct` deploy interface requires an unauthenticated image source link, so that the agent running at ramdisk can download the image from provided link and deploy it to the hard disk.

In a typical deployment, user images are managed by the Image service and in most cases, access is controlled by the Identity service. The `direct` deploy interface relies on the Object Storage service to generate an unauthenticated URL which is accessible for a period (i.e. tempurl).

The problem is the Object Storage service is not always adopted in a deployment due to various reasons, and itself imposes restrictions on deployment. E.g.:

- It has little benefit for a small cloud but takes more hardware resource.
- It requires baremetal nodes to have access to control plane network, which is a restriction to network topology.
- It requires the Image service be configured with a backend of swift, which may conflicts with original one.

As there is no mechanism or means for ironic to leverage a local HTTP server to provide temporary image file for IPA to facilitate a node deployment, this proposal is to offer an alternative by providing such support.

Proposed change

An HTTP server on the ironic conductor node is required for this feature to work.

Currently there are two scenarios, if the `instance_info['image_source']` indicates its a glance image, the `direct` deploy interface generates tempurl via glance client, and stores it to `instance_info['image_url']`, otherwise it will be directly taken as `image_url`. The two cases typically represent using the Bare Metal service in the cloud or as a standalone service, respectively.

The proposal introduces a new string option `[agent]image_download_source` to control which kind of image URL will be generated when the `image_source` is a glance image. Allowed values are `swift` and `http`, defaults to `swift`.

The process of the `direct` deploy interface on different configurations is defined as:

- `swift`: Keeps current logic, generates tempurl and update it to `instance_info['image_url']`.
- `http`: Downloads instance image via `InstanceImageCache` before node deployment, creates symbolic link to downloaded instance image in the directory accessible by local HTTP service, generates proper URL and updates it to `instance_info['image_url']`.

The existing `[deploy]http_root` is reused for storing symbolic links to downloaded instance images. A new string option `[deploy]http_image_subdir` is introduced to keep it isolated with iPXE related scripts. The default value is `agent_images`. The existing `[deploy]http_url` is reused to generate instance image URLs.

The direct deploy interface will use the same instance cache for image caching, the caching will be performed at AgentDeploy.deploy. After an instance image is cached, the direct deploy interface creates a symbolic link at <http_root>/<http_image_subdir> to reference the instance image. It will be /httpboot/agent_images/<node-uuid> if all goes to default.

The direct deploy interface generates URL for the instance image and updates it to instance_info at AgentDeploy.prepare. The corresponding image URL will be <http_url>/<http_image_subdir>/<node-uuid>. The symbolic link will be removed at AgentDeploy.deploy when a node deploy is done, or AgentDeploy.clean_up when a node is teared down from the state deploy failed.

Rule to convert image

Currently the iscsi deploy interface will convert image to raw if [DEFAULT]force_raw_images is set to True.

While IPA treats instance image in two different ways:

- If the instance image format is raw, stream_raw_images is True and image type is whole disk image, the image will be streamed into the target disk of the Bare Metal.
- Otherwise the image will be cached into memory before written to disk.

To avoid a raw image been cached into the memory of Bare Metal, the direct deploy interface will convert image to raw only if following criteria is met:

- [DEFAULT]force_raw_images is set to True,
- [agent]stream_raw_images is set to True,
- The instance image type is a whole disk image.

The direct deploy interface will recalculate MD5 checksum and update necessary fields to instance_info if image conversion happened.

Cache sharing

iscsi and direct deploy interface are sharing the same cache, but apply different rule to whether the image should be converted to raw. It leads to cache compatibility issue when both interface are in use.

As an example, suppose we deploy node A (using iscsi) with a partition image, then deploy node B (use direct) with the same image. The image in the cache is converted to raw, but according to the rule of direct deploy interface, it assumes image will not be converted to raw, though it specifies force_raw to false to the image cache, due to cache hit, actually no image action will be performed, this will leads to the situation that the direct deploy interface actually provide a raw image but without MD5 recalculation.

Vice versa, if we reverse the order above, the iscsi deploy interface may get a qcow with [DEFAULT]force_raw_images set to true, though its probably not an issue because populate_image will check image format before writing. its still not a consistent behavior.

To address the issue described above, this spec proposes to update ImageCache.fetch_image to take the input argument force_raw into account for the master image file name:

- The master file name is not changed if force_raw is set to False.
- The master file name will have .converted as file extension if force_raw is set to True, e.g.:

```
/var/lib/ironic/master_images/6e2c5132-db24-4e0d-b612-478c3539da1e.
↪converted
```

Note that the `.converted` extension merely acts as an indicator that the image downloaded has gone through the conversion logic. For a raw image in the glance, the name of master image file still has `.converted` as long as `force_raw` argument passed in is True.

Alternatives

Another implementation approach Support all glance backends in the agent is to support IPA directly downloading instance image from glance, the deployment restriction of this approach is the same as agent today, baremetal has to access glance at provisioning network. But it can address the dependency issue on glance backend, thus can be a further work.

Instead of supporting HTTP provisioning from the `direct` deploy interface, it can also be implemented as a new deploy interface, `direct-http` for example.

Data model impact

None

State Machine Impact

None.

REST API impact

None

Client (CLI) impact

None.

ironic CLI

openstack baremetal CLI

RPC API impact

None.

Driver API impact

None.

Nova driver impact

None

Ramdisk impact

None

Security impact

Providing HTTP service on ironic conductor node will expose accessible port thus can be a security impact. There are several ways to improve security:

1. Bind the port to the network interface dedicated to provisioning networks.
2. Configure firewall to prevent access from source IP addresses other than the provisioning networks.

There might be other ways, but thats beyond the scope of this spec.

To allow HTTP server accessing instance image in the cache directory, the file-creation mask of user for ironic conductor service should be configured to be accessible by the user of HTTP service. Most systems use 022 or 002 as the default umask, it should be sufficienth. There would be a security impact if its not the case.

Other end user impact

None

Scalability impact

Instance images will be cached on the ironic conductor node once the `[agent]image_download_source` is set to `http`, it will cost more disk space if the conductor node is using `direct` deploy interface before. The expected space usage basically should be no more than `iscsi` deploy interface.

IPA downloads instance image directly from the conductor node, which will reduce traffic on the control plane network, by the cost of increasing traffic on each conductor node. The consumption should be no more than `iscsi` deloy interface.

Performance Impact

Depending on the hardware and image type, recalculating MD5 checksum for a raw image could consume considerable amount of CPU/IO resources. If the performance on ironic conductor node is in concern, please set `[DEFAULT]force_raw_images` to `False` (The option is `True` by default).

Other deployer impact

When using this feature, an HTTP server should be set up and configured on each ironic conductor node.

Each HTTP servers should be configured to follow symlinks for instance images are accessible from external requests. Refer to `FollowSymLinks` if Apache HTTP server is used, or `disable_symlinks` if Nginx HTTP server is used.

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
kaifeng

Other contributors:
sambetts

Work Items

- Promote instance cache to be a global cache, usable for other interfaces.
- Implement the proposed work for `direct` deploy interface, includes image caching, checksum recalculating, symlink management, etc.
- Update documents.

Dependencies

None

Testing

This feature will be covered by unit test.

Upgrades and Backwards Compatibility

Two new options `[agent]image_download_source` and `[deploy]http_image_subdir` are introduced in this feature.

`[agent]image_download_source` defaults to `swift`, which should have no impact on upgrades.

The change of the cache file naming could probably invalidate some cached instance images after upgrades, they will be re-cached when used, images not referenced will be cleaned up eventually. This will have no impact if caching is disabled before upgrade.

Documentation Impact

Update admin/interfaces/deploy.rst to describe the usage of this feature.

References

None

5.15.107 HTTP(S) proxy support for agent images downloading

https://bugs.launchpad.net/ironic/+bug/1526222

This adds support of proxy configuration for images downloading by agent.

Problem description

Currently Ironic Python Agent (IPA) is able to download images via direct HTTP(S) links, but it does not support proxy configuration. If IPA will support proxy configuration for image downloading user can place caching proxies in the same physical network segments as nodes for reducing owerall network traffic and deploying time. There are two different types of image sources when Ironic does deploy

with IPA: Glance UUID and HTTP(S) URL. When HTTP(s) URLs are used so we can simply utilize HTTP(S) proxy configuration parameter, additional Ironic features are not needed. When we use Glance UUIDs there is a problem with Swift temporary URLs, because current time is used for temporary URLs calculation. In the proxy servers requests with query string parameters are cached separately for each unique query string, therefore if Swift temp URLs are used images can not be cached efficiently on the proxy server side.

Proposed change

Three new optional parameters: `image_http_proxy`, `image_https_proxy` and `image_no_proxy` will be added to agent deploy driver. First two parameters are strings with format PROTOCOL://PROXY_IP:PROXY_PORT. `image_no_proxy` is a list of comma-separated URLs that should be excluded from proxying. New behavior of agent deploy driver methods:

- get_properties() - returns description of new parameters.
- validate() - validate new parameter(s) (if present).
- continue_deploy() - add proxies and no_proxy keys in the image_info dict if parameter(s) present:

```
proxies = {'http': 'http://192.168.0.2:8080',
           'https': 'https://192.168.0.3:4444'}

no_proxy='192.168.1.5,10.0.0.3'
```

If proxies key is present IPA adds a parameter to requests.get() method. Requests library[0] supports no_proxy only as environment variable, not as a get() parameter. If no_proxy parameter is set agent should add it to Pythons os.environ before get() call.

Swift Temporary URL changes:

For caching proxies different URLs are mapped to different files in the cache. Therefore caching of Swift Temporary URLs for images should be implemented on the conductor. When a temporary URL for image is created agent driver stores it into the cache with UUID of Glance image as a key. Agent driver uses URL from cache for same UUIDs and checks expiration of temporary URLs. New integer config parameter `swift_temp_url_cachetime` will be added to `glance` group. If it greater than zero agent driver enables caching of URLs and use its value for new temp URL duration.

Notes about proxy service:

- Proxy should support HTTP/1.1 chunked transfer encoding.
- For SSL image URLs proxy should be configured for termination of SSL connection from client on the proxy side.
- Caching of large files should be enabled on the proxy.

Alternatives

None

[0] http://docs.python-requests.org/en/latest/user/advanced/#proxies

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

N/A

Security impact

Decrypting of HTTPS data on the proxy server side is not recommended for images which contain confidential information.

Other end user impact

None

Scalability impact

Proxy support for image downloading by agent can improve scalability (reduce network traffic and time of deploy) in proper configured environment.

Performance Impact

None

Other deployer impact

- New optional parameters will be added for agent deploy driver in the node.driver_info: `image_http_proxy`, `image_https_proxy`, `image_no_proxy`.
- A new config option `swift_temp_url_cachetime` will be added in `glance` group.
- Deployer must install and configure proxy service(s).

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
yuriyz

Work Items

- Implement proxy parameters for IPA deploy driver.
- Implement Swift Temporary URLs cache.
- Add unit tests.

Dependencies

None

Testing

Unittests will be added.

Upgrades and Backwards Compatibility

None

Documentation Impact

Usage of agents proxy configuration will be documented.

References

5.15.108 Allocation API

https://storyboard.openstack.org/#!/story/2004341

This spec proposed creating of API for *allocation* of nodes for deployment.

Problem description

The users of standalone ironic do not have an out-of-box means to find a suitable node to deploy onto. The metalsmith project was created to add this gap short-term, but it is not suitable for consumer code that is not written in Python. A potential consumer is a K8S provider for standalone ironic.

The API user story is as follows:

> Given a resource class and, optionally, a list of required traits, return me an available bare metal node and set `instance_uuid` on it to make it as reserved.

Proposed change

Overview

This RFE proposed a new ReST API endpoint `/v1/allocations` that will initially allow creating and deleting *Allocation* resources.

Two implementations of the allocation process are planned:

1. Via the database, similar to how metalsmith now operates.
2. Via the Placement service, similar to how nova currently operates.

This spec concentrates on the API design and the first (standalone) case.

Allocation process

An allocation happens as follows:

1. An API client sends a `POST /v1/allocations` request, specifying a resource class, and optionally traits and node UUID.
2. The allocation request is routed to a random available conductor.
3. The conductor creates an *Allocation* object in the database with `state=allocating` and `conductor_affinity=<host name>`.
4. A thread is spawned for the remaining allocation process, and the allocation object is returned to the caller.

Allocation: database backend

The following actions are done by the conductor handling the allocation when database is used as backend (the only option in this spec):

1. Fetch list of nodes from the database with:
   - `provision_state=available`
   - `maintenance=False`
   - `power_state!=None`

     Note

     This is required for compatibility with really old API versions that allow creating nodes directly in the `available` state.

   - `instance_uuid=None`
   - `resource_class=<requested resource class>`
   - `uuid` in the list of candidate nodes (if provided)

- requested traits are a superset of node traits

2. If the list is empty, set allocations `state` to `error` and `last_error` to the explanation.
3. Shuffle the list, so that several processes do not try reserving nodes in the same order.
4. Acquire a lock on the first node. If locking succeeds, check that the assumptions are still valid about this node, and reserve it by setting its `instance_uuid` to the `uuid` of the allocation. In the same database transaction:
   - set allocations `node_id` to the nodes ID,
   - set allocations `state` to `active`,
   - set nodes `allocation_id` to the allocations ID,
   - add matched traits to nodes `instance_info`.

   Note

   Since the conductor may not have the hardware type for the selected node, we will update TaskManager to avoid constructing the `driver` object.

5. If something fails on the previous step, proceed to the next node. If no more nodes are left, set the allocations `state` to `error` and `last_error` to the explanation.

Deallocation: database backend

The deallocation process will in one transaction:

- unset nodes `instance_uuid`,
- unset nodes `allocation_id`,
- delete the allocation.

The deallocation is triggered either explicitly via API or when a node is torn down (at the same time when nodes `instance_uuid` and `instance_info` are purged).

Note

In the future we might consider supporting *sticky allocations* which survive nodes tear down. This is outside the scope of this spec.

There is one important difference between using just `instance_uuid` and using the allocation API: `instance_uuid` can be set and unset for `active` nodes, while for allocations it will be forbidden. The reason is that with the future Placement backend removing an allocation would mark the node as available in Placement.

HA and take over

- When a conductor restarts, it fetches allocations with
  - `conductor_affinity=<host name>`
  - `state=allocating`

and starts the allocation procedure for each of them.

- If the conductor handling an allocation stops without a replacement, the reservation becomes orphaned. All conductors periodically fetch list of allocations belonging to dead conductors and each tries to resume them.

  First, it tries to update the `conductor_affinity` by doing something like:

  ```
  UPDATE allocations SET conductor_affinity=<new host name>
     WHERE id=<allocation ID> AND conductor_affinity=<dead host name>
  ```

  If the query updated 1 row, we know that the new conductor now manages the allocation. Otherwise we know that another conductor took it over.

- To avoid rare races with this take over procedure, the normal update will also look like:

  ```
  UPDATE allocations SET <new values>
      WHERE id=<allocation ID> AND conductor_affinity=<current host name>
  ```

Alternatives

- Make each consumer invent their own allocation procedure or use metalsmith.
- Write a new service for managing reservations (probably based on metalsmith code base).
- Make the API blocking and avoid having states for allocations. Such an approach would result in easier API and implementation, but it can be problematic when using an external system, such as Placement, as a backend, since the requests to it make block the RPC thread.

  Additionally, the asynchronous design will make it easier to introduce a bulk allocation API in the future, if we decide so.

Data model impact

Introduce a new database/RPC object *Allocation* with the following fields:

- `id` internal integer ID, not exposed to users.
- `uuid` unique UUID of the allocation.
- `name` unique name of the allocation, follows the same format as nodes names.

  Note

  This field is useful, for example, for systems using host names, like metalsmith.

- `node_id` reserved node ID (can be `null`) - foreign key to the `nodes` table.
- `updated_at`/`created_at` standard update/creation date time fields.
- `resource_class` mandatory requested resource class.
- `candidate_nodes` list of node UUIDs to choose from (can be `null`).

> **Note**
>
> This allows a caller to pre-filter nodes by arbitrary criteria.

- `state` allocation state, see *State Machine Impact*.
- `last_error` last error message.
- `conductor_affinity` internal field, specifying which conductor currently handles this allocation.

Introduce a helper table `allocation_traits` mapping an allocation to its requested traits (very similar to `node_traits`).

Update the `nodes` table with a new foreign key `allocation_id`. It will be set to a ID of a corresponding allocation. Unlike `instance_uuid`, it will only be set when an allocation is created. If `allocation_id` is not empty, `instance_uuid` will hold the UUID of the corresponding allocation (the opposite is not necessary true).

State Machine Impact

No impact on the node state machine.

This RFE introduces a simple state machine for *Allocation* objects, consisting of three states:

- `allocating` allocation is in progress (initial state).
- `active` allocation active.
- `error` allocation failed.

In the initial version only the following paths are possible:

- from `allocating` to `active` on success.
- from `allocating` to `error` on failure.

In the future we may allow moving from `error` back to `allocating` to retry the allocation.

REST API impact

- `POST /v1/allocations` create an allocation.

  The API accepts a JSON object. The following field is mandatory:

  - `resource_class` requested nodes resource class.

  The following fields are accepted:

  - `uuid` to create an allocation with the specific UUID.
  - `candidate_nodes` to limit the query to one of these nodes.

    > **Note**
    >
    > This value is converted from names to UUIDs internally.

  - `traits` list of requested traits.

  - `name` allocation name.

  The normal response is HTTP CREATED with the response body being the created allocation representation. An allocation is created in the `allocating` state.

  Error codes:

  - 400 Bad Request if
    - any node from `candidate_nodes` cannot be found (by name or UUID).
    - the `resource_class` value is invalid.
    - `traits` is provided and is not a list of valid trait strings.
    - `name` is not an accepted name.
  - 406 Conflict if
    - the provided `uuid` is not unique or matches `instance_uuid` of any node.
    - the provided `name` is not unique.

- `GET /v1/allocations` list allocations.

  Parameters:

  - `fields` list of fields to retrieve for each allocation.
  - `state` filter allocations by the state.
  - `resource_class` filter allocations by resource class.
  - `node` filter allocations by node UUID or name.

  Error codes:

  - 400 Bad Request if
    - `state` is invalid.
    - `resource_class` is invalid.
    - `node` does not exist.
    - any of the requested fields is invalid.

- `GET /v1/allocations/<uuid or name>` retrieve an allocation.

  Parameters:

  - `fields` list of fields to retrieve.

  Error codes:

  - 400 Bad Request if any of the requested fields is invalid.
  - 404 Not Found if the allocation is not found.

- `DELETE /v1/allocations/<uuid or name>` remove the allocation and release the node.

  No request or response body. Response code is HTTP 204 No Content.

  Error codes:

  - 404 Not Found if the allocation is not found.

  - 409 Conflict if the corresponding node is `active` or is in a state where updates are not allowed.

> **Note**
>
> This request will succeed only for real allocations. It will not be possible to unset `instance_uuid` created without an allocation (i.e. by direct `PATCH` to a node) using this API.

- `GET /v1/nodes/<node UUID or name>/allocation` get allocation associated with the node.

  Parameters:

  - `fields` list of fields to retrieve.

  The response body is the *Allocation* object representation.

  Error codes:

  - 404 Not Found if
    * the node cannot be found.
    * there is no allocation for the node.
  - 400 Bad Request if
    * node has `instance_uuid` that does not correspond to any allocation.
    * any of the requested fields is invalid.

- Update `GET /v1/nodes`, `GET /v1/nodes/detail` and `GET /v1/nodes/<node UUID or name>`:

  Expose the new `allocation_uuid` field (converted from the nodes `allocation_id`).

- Update `PATCH /v1/nodes/<node UUID or name>`:

  If `instance_uuid` is unset and the current value corresponds to an allocation:

  - if node is `active` or in a state that disallows updates, and `maintenance` mode is off, return HTTP 409 Conflict,
  - otherwise delete the allocation.

  If `instance_uuid` is set, do NOT create an allocation, keep the previous behavior.

> **Note**
>
> This is needed to avoid affecting the nova virt driver. This decision may be revisited in future API versions.

  The `allocation_uuid` field is read-only, an attempt to change it directly will result in HTTP 400 (Bad Request).

- Update `DELETE /v1/nodes/<node UUID or name>`:

  If a node is deleted with allocation (possible only in maintenance mode), the allocation is deleted as well.

Client (CLI) impact

ironic CLI

None.

openstack baremetal CLI

The matching commands will be created:

```
openstack baremetal allocation create --resource-class <class> \
   [--trait <trait>] [--trait <trait>] [--uuid <uuid>] [--name <name>]
openstack baremetal allocation list [--state <state>] [--fields <fields>]
   [--resource-class <class>] [--node <UUID or name>]
openstack baremetal allocation get <uuid or name> [--fields <fields>]
openstack baremetal allocation delete <uuid or name>
```

The `allocation_uuid` field will be exposed.

RPC API impact

Two new RPC calls are introduced:

- Creating an allocation

```
def create_allocation(self, context, allocation):
   """Create an allocation.

   Creates the provided allocation in the database, then starts a thread
   to process it.

   :param context: context
   :param allocation: allocation object.
   """
```

- Deleting an allocation

```
def destroy_allocation(self, context, allocation):
   """Destroy an allocation.

   Removes the allocation from the database and releases the node.

   :param context: context
   :param allocation: allocation object.
   """
```

metalsmith impact

The metalsmith project implements a superset of the proposed feature on a client side. After this API is introduced, metalsmith will switch the `reserve_node` call to using it in the following way:

- If the request contains a `resource_class` and, optionally, `traits` and candidate nodes, the new API will be used.

- If the request contains anything not supported by the new API, metalsmith will continue client-side node filtering, and will create an allocation with a list of suitable nodes.

Driver API impact

None

Nova driver impact

None

In the future we may use the allocation API in the nova driver, but there are no plans for it now. Currently going through the allocation API will result in an attempt of double allocation in Placement if Placement is used as an allocation backend.

Ramdisk impact

None

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

A new periodic task will run on each conductor to periodically check for stalled reservations belonging to dead conductors. The default period will be 60 seconds. It will be possible to disable it, in which case the allocations may get stuck forever if their assigned conductor dies.

Other deployer impact

None

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
dtantsur

Work Items

- Add new tables and the *Allocation* RPC object.
- Add RPC for allocating/deallocating.
- Add API for allocations creation and deletion, and API reference.
- Update conductor starting procedure to check for unfinished allocations.
- Add a periodic task to check for orphaned unfinished allocations.

Dependencies

None

Testing

- Unit tests and Tempest API will be provided.
- The standalone integration tests will be updated to use the new API.
- We can add support for the new API to bifrost (e.g. via metalsmith), and test it in a bifrost CI job.

Upgrades and Backwards Compatibility

This change is fully backward compatible. Code using `instance_uuid` for allocations is not affected.

Documentation Impact

API reference will be provided.

References

5.15.109 Anaconda deploy interface

https://storyboard.openstack.org/#!/story/2007839

Problem description

We would like Ironic to provide the following features (described in more detail later in this spec):

- Create LVM logical volumes, volume groups, and physical volumes
- Create MD raid arrays
- Create Linux filesystems
- Generate Linux fstab files based on created Linux filesystems

Support for creating LVM and MD structures has significant usefulness in when provisioning Nodes with multiple physical storage devices.

Use-cases include:

- A user wants to use a LVM or MD feature
- A user wants to create a Linux filesystem on any block device created during provisioning
- A user wants an fstab with entries for all filesystems created by the Ironic deploy interface

While Ironic supports creation of custom MD raid arrays using deploy templates, it could lead to an explosion of flavors[1].

Proposed change

This spec suggests implementing these block device provisioning features by creating a new anaconda deploy interface using Anaconda. The anaconda deploy interface allows a user to select one of a set of pre-defined Kickstart configurations[2].

The primary advantage of this approach is in allowing customization of the deployed operating system with all features offered by Kickstart commands.

This spec is interested in the following Anaconda features. In most cases, each feature described has a 1:1 relationship with a Kickstart command:

Feature	Ironic	Anaconda
Zeroize GPT/MBR structures	yes	yes
Create a partition in GPT/MBR	yes	yes
Install a bootloader (GRUB)	yes	yes
Create a LVM logical volume	no	yes
Create a LVM volume group	no	yes
Create a LVM physical volume	no	yes
Create a MD raid array	limited	yes
Create filesystems specified by the user	no	yes
fstab entry generator	no	yes

As shown, there are several block device-related features supported by this spec.

In addition, there are some desired pseudo-features, for example LVM physical volume on MD raid array that are accomplished by the user providing sequences of (imperative) Kickstart commands that effectively implement this.

Deploying a node with `Anaconda` deploy driver involves solving following problems:

1. Identifying the correct Anaconda version to use with the OS image.
2. Getting user-provided kickstart file to the Anaconda runtime.
3. The Operator/User generating an OS image with necessary tools in a format supported by Anaconda.
4. Passing correct kernel cmdline arguments to Anaconda by generating them in PXE boot driver.
5. Sending status back to Ironic API when the deployment starts/ends or crashes.
6. Handling cleaning when Anaconda deploy driver is used.

Scope

The scope of the Anaconda deploy interface is limited to

- CentOS/RHEL >= 7 and Fedora >=31
- Anaconda will be used to deploy the OS image.

[1] https://specs.openstack.org/openstack/ironic-specs/specs/approved/deploy-templates.html#current-limitations
[2] https://pykickstart.readthedocs.io/en/latest/kickstart-docs.html

- Support both UEFI and Legacy BIOS mode.

Matching the OS image with correct anaconda version

Anaconda is a two stage installer where the ramdisk is considered stage1. Once stage1 is loaded, it tries to download stage2 of the installer over the network. Stage2 is a squashfs[3] image and the location of stage2 can be specified using `inst.stage2` kernel command line argument.

```
inst.stage2=http://<address>:<port>/httproot/<node-uuid>/squashfs.img
```

To deploy the OS using anaconda, apart from the OS image,kernel,ramdisk(stage1) and anaconda squashfs(stage2) images are required. All these artifacts should be uploaded to glance and associated with the OS image by the operator.

```
openstack image create --disk-format raw --container-format compressed \
    --file path/to/os_image.tar.bz2 \
    --property kernel_id=$MY_VMLINUZ_UUID \
    --property ramdisk_id=$MY_INITRD_UUID \
    --property stage2_id=$MY_ANACONDA_SQUASHFS_UUID \
    --property os_distro=RHEL \
    --property os_version=7 centos-7-base \
    --property ks_template=glance://uuid``
```

This is a departure from how direct deploy interface works where the kernel_id and ramdisk_id are either in configuration file or set in driver_info. IPA is distro agnostic, Anaconda is not. There is no single Anaconda installer version that is compatible with all major versions of CentOS. Each major version of CentOS has its own version of anaconda installer. For this reason we require the operator to associate correct PXE kernel, PXE ramdisk and Anaconda squashfs with the OS image as properties in Glance. The Anaconda deploy interface shall validate the image properties and make sure that all required properties are set before the deployment.

Kickstart templates

Anaconda installer needs a kickstart file to deploy an operating system non-interactively. The kickstart file is used to automate the deployment of the operating system. The default kickstart template when not modified by the operator should automatically partition the available disks using the autopart mechanism[4] and deploy the OS. The default kickstart file template will be named default_kickstart.template and referenced by the configuration option `default_ks_template` in ironic.conf under `[kickstart]` section.

Example default kickstart template:

```
lang en_US
keyboard us
timezone America/Los_Angeles --isUtc
#platform x86, AMD64, or Intel EM64T
text
install
cmdline
reboot
```

(continues on next page)

[3] https://en.wikipedia.org/wiki/SquashFS
[4] https://pykickstart.readthedocs.io/en/latest/kickstart-docs.html#autopart

(continued from previous page)

```
selinux --enforcing
firewall --enabled
firstboot --disabled
auth --passalgo=sha512 --useshadow


bootloader --location=mbr --append="rhgb quiet crashkernel=auto"
zerombr
clearpart --all --initlabel
autopart
```

All kickstart templates will be automatically appended with following mandatory sections during deployment

```
# Downloading and installing OS image using liveimg section is mandatory
liveimg --url={{ liveimg_url }}

# Following %pre, %onerror and %trackback sections are mandatory
%pre
/usr/bin/curl -X PUT -H 'Content-Type: application/json' -H 'Accept:␣
↪application/json' -d '{.."agent_status": "start"}' http(s)://host:port/v1/
↪heartbeat/{{node_ident}}
%end

%onerror
/usr/bin/curl -X PUT -H 'Content-Type: application/json' -H 'Accept:␣
↪application/json' -d '{"agent_status": "Error: Deploying using anaconda.␣
↪Check console for more information."}' http(s)://host:port/v1/heartbeat/{
↪{node_ident}
%end

%traceback
/usr/bin/curl -X PUT -H 'Content-Type: application/json' -H 'Accept:␣
↪application/json' -d '{.."agent_status": "Error: Anaconda crashed␣
↪unexpectedly."}' http(s)://host:port/v1/heartbeat/{{node_ident}
%end

# Sending callback after the installation is mandatory
%post
/usr/bin/curl -X PUT -H 'Content-Type: application/json' -H 'Accept:␣
↪application/json' -d '{.."agent_status": "success"}' http(s)://host:port/v1/
↪heartbeat/{{node_ident}
%end
```

Multiple %pre, %post, %traceback and %error sections can exist in a kickstart file. These sections will be processed and executed in the order they are encountered[5].

Custom kickstart templates should be uploaded to glance or hosted in a webserver accessible by the conductor or on the conductors filesystem.

[5] https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/installation_guide/sect-kickstart-syntax#sect-kickstart-preinstall

The operator can set the kickstart file using URI formats `glance://<uuid>` or `http(s)://host:port/path/ks.cfg` or `file://path/to/ks.cfg`

If the API user decides to store the kickstart file in glance they can do so by running the following command

```
openstack image create --file ks.cfg --container-format bare \
    --disk-format raw custom_kickstart_template
```

Users can specify a specific kickstart template for a node via the nodes instance_info field, with key ks_template. For example:

```
openstack baremetal node set $NODE_UUID \
    --instance_info ks_template=glance://uuid

or

openstack baremetal node set $NODE_UUID \
    --instance_info ks_template=http(s)://port:host/path/ks.cfg

or

openstack baremetal node set $NODE_UUID \
    --instance_info ks_template=file://path/to/ks.cfg
```

The user can also associate a kickstart template with an OS image(image_source) in glance. The template specified in the instance_info will take precedence followed by `ks_template` property on the OS image. Finally if `ks_template` property is not present in both instance_info and OS image then the default kickstart template specified in the configuration file will be used.

The custom kickstart template will be downloaded and stored in httproot/<node-uuid>/ks.cfg. Where as `httproot` is defined in `http_root` configuration item under `[deploy]` section of `ironic.conf` configuration file. Once the custom kickstart template is downloaded it will be validated against `os_distro` and `os_version`

```
ksvalidator -v RHEL7 ks.cfg
```

os_distro and os_version are properties of image_source(The OS image). The API user is required to set os_distro and os_version. If there is no os_distro or os_version set on the image_source, the kickstart file will be validated against `DEVEL` version of kickstart syntax. See `ksvalidator -l` for list of supported kickstart versions[6].

The OS_DISTRO should be one of RHEL and OS_VERSION should be either 7 or 8.

The kickstart file is passed to anaconda installer using the kernel cmdline argument `inst.ks`

```
inst.ks=http(s)://<address>:<port>/httproot/<node-uuid>/ks.cfg
```

Kernel command line arguments

Two important kernel command line arguments are required for the anaconda installer to work.

1. `inst.stage2`
2. `inst.ks`

[6] https://pypi.org/project/pykickstart/

Both of these kernel command line arguments will be appended to `pxe_options` dictionary. A function similar to get_volume_pxe_options() will be added to pxe_utils to facilitate this.

OS image format

While Anaconda supports installing individual RPM packages from a remote server, the deployment driver will only support installation of disk image formats described by liveimg[7]. liveimg accepts tarballs, squashfs images and any mountable disk images. Users can generate squashfs images and tarballs.

Deployment status

Anaconda installer doesnt know how to talk to Ironic APIs. However we can have %pre and %post sections of kickstart file make API calls to Ironic. The ramdisk will use heartbeat API to talk to the Ironic API. The %pre %onerror, %traceback, and %post sections will be populated with `curl` calls to heartbeat API when conductor renders the kickstart template.

The %pre and %post sections of kickstart files are executed in order they are encountered by anaconda.

At the start of the installation following status will be sent using %pre section of the kickstart file from anaconda ramdisk to lookup

POST {callback_url: , agent_token: <token>, agent_version: ,

agent_status: start}
http(s)://<address>:<port>/v1/heartbeat/{{node_ident}}

On receiving the start status from anaconda ramdisk, the conductor will set driver_internal_info[agent_status] = start

At the end of the OS installation %post section will be used to send following message back to Ironic

POST {callback_url: , agent_token: <token>, agent_version: ,

agent_status: success}
http(s)://<address>:<port>/v1/heartbeat/{{node_ident}}

On receiving the success the conductor will move the Ironic node to active state depending on the current state and set driver_internal_info[agent_status] = success

If there are errors during installation we will capture those error using %onerror[8] and %traceback[9] sections of kickstart file, then send the error status to Ironic

POST {callback_url: , agent_token: <token>, agent_version: ,

agent_status: Error: <msg>}
http(s)://<address>:<port>/v1/heartbeat/{{node_ident}}

On receiving the Error status the conductor will set the provision_state of Ironic node to deploy_failed depending on the current status of the node and set the last_error field of the Ironic node.

There will be no calls to /v1/lookup API from ramdisk. The agent token will be generated when the kickstart file is rendered. Agent token will be embedded in the kickstart file for the heartbeat `curl` call to use. The driver avoids calls to lookup API because it is difficult to read and extract agent token using scripts in the ramdisk.

[7] https://pykickstart.readthedocs.io/en/latest/kickstart-docs.html#liveimg
[8] https://pykickstart.readthedocs.io/en/latest/kickstart-docs.html#chapter-7-handling-errors
[9] https://pykickstart.readthedocs.io/en/latest/kickstart-docs.html#chapter-8-handling-tracebacks

Cleaning

The `PXEAnacondaDeploy` driver will inherit from `AgentBaseMixin` interface and `DeployInterface` similar to `PXERamdiskDeploy` driver. This implies that the cleaning will be done by the agent Driver not by the Anaconda deploy driver.

During deployment the `PXEAnacondaDeploy` driver will use the properties associated with the `image_source` to figure out the deploy_kernel and deploy_ramdisk. However during cleaning it will use the driver_info[deploy_kernel] and driver_info[deploy_ramdisk] fields to determine the cleaning kernel and ramdisk. This mean the driver_info deploy_* fields should refer to IPA kernel/ramdisk. This change is likely to be a point of confusion.

Alternatives

The Anaconda deployment driver is specific to Red Hat based distributions. This deployment driver wont support distributions not supported by Anaconda. For example ubuntu is not supported by this deploy driver. A similar driver can be implemented to support ubuntu using preseed[10] files.

Another alternative is to define a generic partition configuration format and use that configuration instead of kickstart file. This new generic partition configuration will be validated by the conductor and sent to Ironic python agent during deployment. IPA will read the generic partition configuration and use libraries like blivet[11] to partition/format the disks. With this approach the deploy driver isnt tied to a specific distribution or vendor. We explored this approach but we found it to be too complex and reinventing lot of things the distribution installers already do.

The kickstart file is either uploaded to glance or hosted in a webserver in current proposal. Alternatively we can add a new field named `kickstart` to `nodes` table which accepts raw kickstart file

```
openstack baremetal rebuild --kickstart path/to/ks.cfg <node-uuid>
```

Data model impact

None

State Machine Impact

None

REST API impact

Add an optional field `agent_status` to v1/heartbeat API, which can be used to receive deployment status from the anaconda deploy driver.

POST {.. agent_status: <status>} /v1/heartbeat/{{node_ident}}

Client (CLI) impact

None

[10] https://wiki.debian.org/DebianInstaller/Preseed
[11] https://pypi.org/project/blivet/

openstack baremetal CLI

None

openstacksdk

None

RPC API impact

RPC API needs to be updated to handle the new `agent_status` in heartbeat API.

Driver API impact

None

Nova driver impact

There is no impact to novas Ironic driver at this time.

Ramdisk impact

There is no impact to Ironic-python-agent.

Security impact

The `heartbeat` method implemented by the driver has to be unauthenticated so that anaconda can POST to the status API without a token. An attacker could potentially cause targeted denial of service attack by sending invalid/incorrect status to Ironic nodes since the API is unauthenticated. This issue is mitigated by mandatory agent token verification.

Other end user impact

An OS image that can be deployed via liveimg kickstart command should be uploaded to glance along with relevant anaconda installers PXE kernel, ramdisk and squashfs image. The PXE kernel,ramdisk and squashfs need to be associated with the OS image.

```
openstack image set IMG-ID --property kernel_id=$MY_VMLINUZ_UUID \
    --property ramdisk_id=$MY_INITRD_UUID --property \
    squashfs_id=$MY_ANACONDA_SQUASHFS_UUID
```

The end user can make use of their custom kickstart templates during deployment by working with the Operator. The Operator can set the instance_info `ks_template` key with the path of user provided kickstart template. The kickstart template can be in glance `glance://uuid`, webserver `http(s)://host:port/path/ks.cfg` or on the filesystem `file://etc/ironic/ks.cfg` of the conductor.

```
openstack baremetal node set $NODE_UUID --instance_info ks_template=<TMPL>
```

Scalability impact

None

Performance Impact

None

Other deployer impact

The operator has to set default kickstart template under `[kickstart]` section of Ironic configuration file.

```
[kickstart]
default_ks_template=$Ironic_CONF_DIR/kickstart/default_ks.template
```

The `kickstart` deploy interface must be set on the node .. code:: bash

    openstack baremetal node set <NODE> deploy-interface kickstart

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
zer0c00l, sagarun@gmail.com

Work Items

1. Definition of default kickstart template and configuration items related to kickstart deploy template.
2. Implementation of core deploy driver that fetches artifacts from glance, generates PXE configuration files, renders kickstart templates into httproot
3. A CI job to test the anaconda deploy driver
4. Documentation for operators and users

Dependencies

None

Testing

- This driver should be testable in gate. Enhancements might be needed to gate to get this working.
- Devstack support will be added for this driver so that it can be tested easily.

Upgrades and Backwards Compatibility

None

Documentation Impact

Clear operator and user documentation need to be added on kickstart deploy interface and how to make use of it.

References

5.15.110 Promote Ansible deploy interface to ironic

https://blueprints.launchpad.net/ironic/+spec/ansible-deploy-driver

This spec presents a new deploy driver interface for provisioning nodes via Ansible playbooks.

Problem description

Sometime it may be required to treat (some) baremetal nodes more like pets than cattle, providing per-node custom logic of deploy process.

Currently such customization in Ironic is not an easy task, as developer/cloud administrator would have to do one or more of the following:

- Modify the code of Ironic deploy driver.
- Modify the code of Ironic Python Agent.
- Modify the code of ironic-lib.
- Rebuild IPA ramdisk.
- Upload new ramdisk to Glance or HTTP server(s).
- Update the nodes info with new ramdisk.
- Update deploy driver on Ironic conductors.
- Restart conductors.

This problem can be partially solved in deploy driver based on external templates for a configuration management tool.

Possible use cases and advantages:

- Custom actions with hardware at any stage of deploy process with vendors utilities, in-band or out-of-band.
- Easy replace usage of one Linux utility with other for deploy.
- Deep tuning of deploy process.
- Changing behavior of deploy process without restart the conductors.
- Long life time of deploy ramdisk - vendors utilities can be downloaded from external sources during deploy process.
- Separation of concerns - ironic manages the what part, Ansible the how part
- Allows BMs to be treated more like pets

Proposed change

This spec proposes such deploy interface based on Ansible[1] as configuration management solution. Effectively it uses ironic for its power, management and boot capabilities, and shifts the logic of deployment itself to Ansible playbooks.

Such deploy interface is already implemented and is available as part of `ironic-staging-drivers` project[2]. It can be used as possible deploy interface for `ipmi` hardware (and possibly other hardware as well). Therefore this spec is proposing promotion of the `ansible` deploy interface for inclusion in the core ironic as one of available deploy interfaces.

Below is a short description of this deploy interface architecture and current capabilities. More information is available in the `ironic-staging-drivers` project documentation at[3].

Use case

This deploy interface is mostly suitable for undercloud-like or standalone ironic usage, where the operator of ironic service is usually the owner of the images to be deployed and/or the deployed instances themselves. It does however already include set of Ansible playbooks that try to mimic the capabilities of the standard `direct` deploy interface of ironic as close as possible.

This deploy interface is really useful when, during provisioning, there is a need to perform some low-level system configuration that would be hard to do in the already provisioned instance (like placing the root partition on a LVM/software RAID) or would require an extra reboot (like changing kernel parameters).

General overview

We chose Ansible for the following reason:

- open source (GPLv3 + parts as MIT), mature and popular enough, including OpenStack ecosystem itself
- written and extendable in Python, fits nicely in the OpenStack ecosystem
- configuration files are human-readable YAML
- agent-less by design, with minimal requirements on managed node, only Python and SSH server are required.

There are two ways to utilize Ansible from Pythons program: use Ansible Python API or run CLI utility with parameters. Ansible Python API currently does mandatory fork of callers process, this behaviour is not suitable for Ironic conductor (oslo.messaging does not allow forks at least). Besides, Ansible licensing choice (GPLv3) prohibits usage of Ansible Python API from ironic (Apache 2.0). Therefore `ansible-playbook` CLI utility is used.

Each action (deploy, clean) is described by a single Ansible playbook with roles, which is run as a whole during deployment, or tag-wise during cleaning. Control of cleaning steps is through Ansible tags and auxiliary clean steps file. The playbooks for actions can be set per-node, as is cleaning steps file.

The dreploy interface tries to reuse as much code from ironic as possible, and interface-wise is quite similar to the `direct` deploy interface.

Currently this interface supports two modes for continuing deployment or cleaning:

- having the deploy ramdisk calling back to ironic APIs heartbeat endpoint (default)

[1] https://www.ansible.com/
[2] https://opendev.org/x/ironic-staging-drivers/
[3] http://ironic-staging-drivers.readthedocs.io/en/latest/drivers/ansible.html

- polling the node until the ssh port is open as part of a playbook

We propose to remove the latter from the interface when moving it to ironic, as support for it is not tested in gates, it decreases performance due to polling, and in general makes code more complicated.

A custom Ansible callback plugin is used for logging. It can read the logging configuration from ironic configuration file, and thus emit log entries for Ansible events directly to the logging backend ironic is also using (works best with journald backend).

Deploy

The interface prepares a set of parameters which are needed for deploy, including access info for the node to be deployed. It then executes the `ansible-playbook` script passing it all the collected information, with node access info being used to register the node in Ansible inventory at runtime.

Supported image types are whole-disk images and partition images with local boot option, **netboot is currently not supported**. Compressed images are downloaded to deploy ramdisk and converted to actual disk device/partition, RAW images are streamed to the target directly.

Creating a configdrive partition is supported for both whole disk and partition images, on both msdos and GPT labeled disks.

Root device hints are currently supported in their basic form only (with exact matches, without oslo.utils operators), there are patches to add full support on review. If no root device hint is provided for the node, first device returned as part of `ansible_devices` Ansible fact is used as root device to create partitions on or write the whole disk image to.

Cleaning

Cleaning procedure for Ansible deploy interface:

- Each cleaning step is a tag in the Ansible playbook file used for cleaning. Available steps, their priorities and corresponding tags are defined in a auxiliary cleaning steps configuration file.
- `get_clean_steps()` method returns a list of cleaning steps defined in mentioned configuration file.
- `prepare_cleaning()` method loads the same ramdisk as for deploying.
- `execute_clean_step()` method does synchronous execution of cleaning step via Ansible, executing only tasks with Ansible tags assigned to the cleaning step.

Default cleaning playbook provided with the interface supports `erase_devices_metadata` and `erase_devices` clean steps of `direct` deploy interface by executing shallow disk metadata cleanup and shredding of disk devices respectively, honoring priorities of those steps set in ironics configuration file.

Alternatives

Use a different deployment customization mechanism or dont support the pet-like treatment.

The short rundown of main pros and cons of current `ansible` deploy interface compared to already available and standard `direct` deploy interface:

- easier to extend for custom provision logic
- is not async
- does not support netboot

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

To support this new deploy interface the deploy ramdisk should include:

- a user with password-less sudo permissions - required
- running SSH server configured for access to this user via SSH keys - required
- Python interpreter (Python2 >= 2.6 or Python3 >=3.5)
  - currently tested with Python2 2.7 only
  - actual Python version required depends on Ansible version used and the version of Python interpreter that executes Ansible on ironic-conductor host
  - current Ansible support of Python3 on managed node is still considered experimental (and is not supported by Ansible 2.1.x at all)
- a software component that upon boot of the ramdisk will make a `lookup` API request to ironic API and make `heartbeat` requests afterwards
  - default choice for such component is IPA
- other system utilities used by deploy or clean playbooks

All or part of those (except the SSH server) can in principle be installed at runtime through additional Ansible tasks in playbooks (reducing the memory footprint and download/boot time of the deploy ramdisk), but also can be provided with deploy ramdisk to shorten the provisioning time.

Re-using IPA as `lookup` and `heartbeat` ironic API client makes it possible to have a unified deploy ramdisk for `direct`, `iscsi` and `ansible` deploy interfaces.

The `ironic-staging-drivers` project includes a set of scripts to perform a simple rebuild of TinyCore Linux based deploy ramdisk (TinyIPA), as well as `diskimage-builder` element to build a suitable ramdisk with this utility, basing it on `ironic-agent` element and thus also containing IPA. Those will be promoted to the new `openstack/ironic-python-agent-builder` project[4].

Rebuild of CoreOS-based deploy ramdisk is currently not supported but such support can be added in the future.

Security impact

Ansible communicates with remote machines over SSH. Deploy process is secure if private SSH key is properly secured (can be accessed only by the user running ironic-conductor service).

Other end user impact

None

Scalability impact

The driver is not async and is blocking. One conductors worker thread per node being provisioned or cleaned is required. Most of the time the thread waits in blocking state for the completion of `ansible-playbook` process. Possible thread pool exhaustion must be accounted for when planning deployment that allows usage of this deploy interface and configuring thread pool size in ironic configuration file (`[conductor]workers_pool_size` option).

There are ideas on how to make the driver async / not blocking, those will be proposed in further specs/RFEs.

Performance Impact

We have conducted some tests to measure performance impact of running multiple `ansible-playbook` processes on ironic-conductor host[5].

The results show that while there is indeed a performance overhead introduced by using the `ansible` deploy interface, it is well within possibilities of quite standard hardware - we were able to provision 50 and 100 nodes concurrently via single ironic-conductor service using this deploy interface, with total provisioning time similar to `direct` deploy interface. Please see the blog post referenced above for mode details on test setup and results.

Other deployer impact

Config options

these are defined in the `[ansible]` section of ironic configuration file

verbosity
: None, 0-4. Corresponds to number of vs passed to `ansible-playbook`. Default (None) will pass vvvv when global debug is enabled in ironic, and nothing otherwise.

[4] https://opendev.org/openstack/ironic-python-agent-builder/
[5] https://pshchelo.github.io/ansible-deploy-perf.html

ansible_playbook_script
Full path to the `ansible-playbook` script. Useful mostly for testing environments when you e.g. run Ansible from source instead of installing it. Default (None) will search in `$PATH` of the user running ironic-conductor service.

playbooks_path
Path to folder that contains all the Ansible-related files (Ansible inventory, deployment/cleaning playbooks, roles etc). Default is to use the playbooks provided with the package from where it is installed.

config_file_path
Path to Ansibles config file. When set to None will use global system default (usually `/etc/ansible/ansible.cfg`). Default is `playbooks_path`/ansible.cfg

ansible_extra_args
Extra arguments to pass to `ansible-playbook` on each invocation. Default is None.

default_username
Name of the user to use for Ansible when connecting to the ramdisk over SSH. Default is ansible. It may be overridden by per-node `ansible_username` option in nodes `driver_info` field.

default_key_file
Absolute path to the private SSH key file to use by Ansible by default when connecting to the ramdisk over SSH. If none is provided (default), Ansible will use the default SSH keys configured for the user running ironic-conductor service. Also note that private keys with password must be pre-loaded into `ssh-agent`. It may be overridden by per-node `ansible_key_file` option in nodes `driver_info` field.

default_deploy_playbook
Path (relative to $playbooks_path or absolute) to the default playbook used for deployment. Default is deploy.yaml. It may be overridden by per-node `ansible_deploy_playbook` option in nodes `driver_info` field.

default_shutdown_playbook
Path (relative to $playbooks_path or absolute) to the default playbook used for graceful in-band node shutdown. Default is shutdown.yaml. It may be overridden by per-node `ansible_shutdown_playbook` option in nodes `driver_info` field.

default_clean_playbook
Path (relative to $playbooks_path or absolute) to the default playbook used for node cleaning. Default is clean.yaml. It may be overridden by per-node `ansible_clean_playbook` option in nodes `driver_info` field.

default_clean_steps_config
Path (relative to $playbooks_path or absolute) to the default auxiliary cleaning steps file used during the node cleaning. Default is clean_steps.yaml. It may be overridden by per-node `ansible_clean_steps_config` option in nodes `driver_info` field.

extra_memory
Memory (in MiB) consumed by the Ansible-related processes in the deploy ramdisk. Affects decision if the downloaded user image will fit into RAM of the node. Default is 10.

post_deploy_get_power_state_retries
Number of times to retry getting power state to check if bare metal node has been powered off after a soft poweroff. Default is 6.

post_deploy_get_power_state_retry_interval

Amount of time (in seconds) to wait between polling power state after triggering soft poweroff. Default is 5.

The last 3 options are effectively copies of similar options in `[agent]` section of configuration file. We could use single option for (some of) those for all deploy interfaces that make use of them, especially if we rename/move them from `[agent]` section to a section with more general name (like `[deploy]`).

Per-node fields in driver_info

These parameters can be provided with driver_info, all are optional and their default values can be set in the ironic configuration file:

ansible_username
User name to use for Ansible to access the node (default is `ansible`).

ansible_deploy_username
Deprecated in favor of `ansible_username`.

ansible_key_file
Private SSH key used to access the node. If none is provided (default), Ansible will use the default SSH keys configured for the user running ironic-conductor service. Also note that private keys with password must be pre-loaded into `ssh-agent`.

ansible_deploy_keyfile
Deprecated in favor of `ansible_key_file`.

ansible_deploy_playbook
Name of the playbook file inside the `playbooks_path` folder to use when deploying this node.

ansible_shutdown_playbook
Name of the playbook file inside the `playbooks_path` folder to use to gracefully shutdown the node in-band.

ansible_clean_playbook
Name of the playbook file inside the `playbooks_path` folder to use when cleaning the node.

ansible_clean_steps_config
Name of the YAML file inside the `playbooks_path` folder that holds description of cleaning steps used by this node, and defines playbook tags in `ansible_clean_playbook` file corresponding to each cleaning step.

Developer impact

Developers may use this deploy interface for drivers.

Implementation

Assignee(s)

Primary assignee:
Pavlo Shchelokovskyy - pas-ha (IRC), pshchelo (Launchpad)

Other contributors:
Yurii Zveryanskyy - yuriyz (IRC), yzveryanskyy (Launchpad)

Work Items

- Copy `ansible` deploy interface from `ironic-staging-drivers` project
  - most changes would happen in imports of unit test modules
- Register the `ansible` deploy interface entrypoint, add it to the list of supported deploy interfaces for `ipmi` hardware type, do not enable it by default in the configuration file.
- Copy documentation.
- Copy the `imagebuild` scripts from `ironic-staging-drivers` project to the new `ironic-python-agent-builder` project
  - Install and use scripts from this new project in DevStack plugins and gate jobs
- Amend ironics DevStack plugin to be able to set up nodes with this deploy interface.
  - Currently will require small rebuild of TinyIPA image during DevStack install.
- Copy and modify the `gate-tempest-dsvm-ironic-staging-drivers-ansible-wholedisk-ubuntu-xenia` gate job, enable it in non-voting mode on ironic project.

Dependencies

Ansible has to be installed on the host running `ironic-conductor` service.

This deploy interface was developed and tested with Ansible 2.1, and targets Ansible >= 2.1 (with some intermediate versions being excluded as incompatible). Currently the gate job testing this deploy interface passes with latest released Ansible version (2.3.2.0 as of this writing).

Also see the *Ramdisk impact* section for changes necessary to the deploy ramdisk.

Testing

- Unit tests are already in place.
- CI testing is already in place
  - as this is a vendor-agnostic deploy interface, it can be tested with virtual hardware in DevStack on upstream gates
  - the job `gate-tempest-dsvm-ironic-staging-drivers-ansible-wholedisk-ubuntu-xenial-nv` is already running in non-voting mode on all changes to `ironic-staging-drivers` project
  - it would have to be copied and modified after appropriate changes are made to ironics DevStack plugin.

Upgrades and Backwards Compatibility

None.

Documentation Impact

Documentation is already available and will have to be moved to ironic code tree as well.

References

5.15.111 Attestation Interface and Keylime Integration

https://storyboard.openstack.org/#!/story/2002713

In order to help verify that baremetal nodes are in a trustworthy state, we are in need of an interface that allows us to take certain actions or verification steps while proceeding along the state machine.

Some of these steps may involve calling an external attestation server, or executing a special step during cleaning in order to ensure that a node is owned by the attestation server.

At a high level, we need an interface of hooks. And there is no better way than to provide a facility to execute external tooling.

Problem description

Terms Glossary

In trying to bring together two unrelated services, a bit of namespace pollution was inevitable. So for those unfamiliar with Keylime terminology and to avoid confusion with Openstack vocabulary, we will define all the terms needed for this spec here.

Trusted Platform Module (TPM) - a microcontroller within a machine which can create and store hashes securely. All nodes looking to use Keylime for attestation will need to have TPM 2.0.

Integrity Measurement Architecture (IMA) - A security subsystem in Linux which gathers hashes of files, file metadata, and process metadata as a measurement of the system. Stores the measurement in the machines TPM. In the context of Ironic and Keylime integration, we will need to run IMA on the node we are attesting.

allowlist - A hash representing the golden state of the node. In the context of Keylime, an allowlist is compared with an IMA measurement to see if the node has been tampered with in an unauthorized way.

Keylime verifier - A component of the Keylime suite which is responsible for comparing the allowlist to the measurement gathered from the node we are attesting. The verifier will run on a machine external to Ironic and the node Ironic is controlling and looking to attest.

Keylime registrar - A component of the Keylime suite which Ironic will need to talk to in order to initiate the attestation workflow for a node. The registrar also runs on a machine external to Ironic and the node. The verifier and registrar may run on the same machine, but it is not necessary and the decision is left to the operator.

Keylime agent - A component of the Keylime suite which runs on the node we are attesting. The agent will command IMA to collect measurements and send the measurements to the verifier.

Keylime tenant - An API which the Ironic conductor will need to use to communicate with the registrar and verifier. Not to be confused with Openstack tenants.

Introduction

Presently, we rely upon a certain level of trust for users that leverage baremetal resources. While we do perform cleaning between deployments, a malicious attacker could potentially modify firmware of attached devices in ways that may or may not be readily detectable.

The solution that has been proposed for this is the use of a measured launch environments with engagement of Trusted Platform Management (TPM) modules to help ensure that the running system profile is exactly as desired or approved, by the attestation service.

But from a security standpoint, security is not always about code. Sometimes security is adherence to process. To leverage TPMs for attestation, we propose Keylime, an open source remote boot attestation and runtime integrity measurement system.

The first step requires a new interface type attestation_interface to be added as a subclass of BaseDriver. This would then come with a attestation_interface implementation which would use Keylime to learn about the security state of a node and manage configurations. All calls to the attestation interface would happen along existing clean and deployment workflows and simply fail transition if a node is deemed to be compromised.

The second step is a set of enhancements for the ramdisk to support TPM 2.0, and installation of the Keylime agent. From there the Keylime agent would communicate with the registrar and verifier. The manager would trigger attestations at certain points along the nodes workflow ex) during the boot process. Note that in order to perform attestation, the verifier must be within the same network as the node.

Proposed change

Attestation Interface

The addition of a `attestation_interface` field in the `nodes` table, which maps to a *task.node.driver.attestation* interface, along with the other standard configuration parameters and defaults behavior that exists with the driver composition model.

Accordingly the `attestation_interface` would be returned on the node object when retrieved via the REST API, and will be able to be set as another interface.

The attestation_interface will provide a means of configuring and orchestrating a nodes connection with a verifier machine.

The Ironic controller will work under the assumption that the network used to communicate with the attestation service is secure and that the attestation entity is also always trustworthy. Trying to concern ourselves with issues like replay attacks or spoofed messages is beyond the scope of IMA attestation workflows.

To accommodate operator workflows wherein an operator may not have access to the attestation service, we cannot allow the attestation service perform any orchestration. This requires all communication to an attestation service to involve the Ironic controller polling an API for a status or instructing the attestation service or node to take action, as opposed to receiving information from the attestation service or node itself. For example, Keylime offers revocation frameworks for taking action immediately upon a node being compromised. However, from Ironics perspective, allowing another service to do any orchestration could put Ironic in a state where it does not know what is happening on the node.

Presently, we are mainly concerned with monitoring deployment and cleaning of a node. The intended workflow will be to use the interface during these steps to ensure the firmware of a node has not been modified.

Keylime Interface

The Keylime interface will inherit from the AttestationInterface class. The purpose of the interface is to allow the controller to gather relevant information about the security state of the node and take action based on the results. Doing so will require methods which will make calls to the Keylime verifier through the available REST API as well as calls to the IPA to pass necessary configuration parameters. Keylime is anticipated to be supported by generic hardware types.

Keylime Configuration

The Keylime verifier and Keylime registrar are two components of the Keylime suite which must be stood up by an operator. The verifier and registrar will need TLS connections over https in order to communicate. The Keylime tenant CLI is installed on ironic controller. The operator will be responsible for securing any network the registrar and verifier are setup in.

Detailed communication requirements are list as following:

> Keylime tenant -> Keylime verifier: mutual TLS connection
>
> Keylime verifier/tenant -> node: unencrypted connection
>
> Keylime verifier/node/tenant -> registrar: mutual TLS connection for post/put requests; unencrypted connection for get/delete requests

Every Keylime agent must have a uuid associated with it in order to register itself with the registrar. It generates its uuid using the Keylime config file. The uuid defaults to a random id.

Allowlist and Excludelist

Allowlists and Excludelists will be generated beforehand and hosted on a remote server or in the conductors filesystem. A filepath for the conductors filesystem or url to a remote server to locate such files will be supplied to Ironic before provisioning. Allowlists may also be signed with a checksum to ensure they have not been tampered with. Such checksums would also be supplied to Ironic with a path or url to the file. Supplying an allowlist is required in order to perform attestation. Excludelists are not required but are used in a majority of Keylime use cases.

The paths of the allowlist, checksum, and excludelist can be saved in `driver_info\keylime_allowlist`, `driver_info\keylime_allowlist_checksum`, and `driver_info\keylime_excludelist`.

Linuxs IMA submodule gathers measurement list signed with TPM quote. The Ironic controller will send the allowlist to the verifier using the Keylime tenant. The Keylime verifier obtains the measurement list and performs attestation by comparing the measurement list against allowlist.

Alternatives

We could add such functionality to various interfaces, but generally attestation will be a specific model for a deployment or portion of a deployment, and thus we may one day have need for vendor specific drivers for particular attestation solutions and workflow. As such, not creating a new interface for this seems less ideal.

Another alternative would be to perform certain checks along state transitions. For example, at clean time we can check the firmware and fail if things have been modified. However, this is undesirable in a scenario where we have strict workflows and processes we want to adhere to. In the situation where an owner lends a node to an untrustworthy lessee the owner might want to ensure the lessee does not perform any unexpected actions. This is also less extensible to other workflows such as a periodic monitoring.

Data model impact

Addition of a `attestation_interface` field to the node object, and this will require a database migration to create the field. The field will default to `None` which will map to a no-attestation interface.

State Machine Impact

No impact to the state machine is expected. All calls to the new interfaces methods will take place in existing workflows driven by the state machine. Action will be taken on a result immediately upon receiving the result.

REST API impact

The `attestation_interface` will be added to the node object and guarded by an API microversion.

Client (CLI) impact

ironic CLI

None

openstack baremetal CLI

The OSC plugin will be changed accordingly to assist users in changing the new `attestation_interface` field.

RPC API impact

This new `attestation_interface` field requires the RPC version to be incremented.

Driver API impact

The attestation interface methods that would be proposed would consist of a `no-attestation` interface defined on a new base class AttestationInterface.

These methods would consist of:

```
def validate_security_status(self, task):
    """Grabs the latest information about the node's security state
    from the attestation machine. Returns nothing on success, raises
    an exception if status is not what we expect or unable to reach
    verifier to obtain a status.
    """

def start_attestation(self, task):
    """Grabs the allowist, allowlist checksum, and excludelist from
    ``driver_info`` instructions. Verifies the integrity of the allowlist
    using the checksum. Attempts to send the allowlist and excludelist to
    the attestation service. Sending allowlist and excludelist allows the
    node to begin attesting itself. Returns nothing on success, raises an
    exception if checksum does not pass or is unable to reach the
    verifier to send allowlist/excludelist.
    """

def unregister_node(self, task):
    """Unregisters the node from the verifier machine. Returns
    nothing on success, raises an exception if status is not what
```

(continues on next page)

(continued from previous page)

```
    we expect.
    """
```

These methods can be used during the nodes cleaning and deployment time. The action taken on a particular security state will be configurable. Whether or not we raise an error on attestation failure will be configurable.

A few additional variables will need to be saved as part of `driver_info` in order to manage the node. These include:

`driver_info\keylime_allowlist` the allowlist for a node.

`driver_info\keylime_allowlist_checksum` a checksum for the allowlist to ensure the allowlist has not been tampered with.

`driver_info\keylime_excludelist` the excludelist for a node.

`driver_info\keylime_agent_uuid` the uuid for a Keylime agent. Needed for querying the verifier for a security status and associating an allowlist/excludelist pair with a node in the Keylime verifier.

Workflow

With all this in mind, we have devised the following workflow for deployment/ cleaning using a Keylime implementation of the attestation interface.

Beforehand, the operator will stand up a machine with the Keylime verifier and registrar. The user will generate their own allowlist, allowlist checksum, and excludelist for the node. An admin may make these files available on the same machine as the Ironic controller and pass in the filepath to `driver_info` or a non admin may make these files available to grab and instead pass in a url to `driver_info`. This step must be done before provisioning. The operator will also pass in how to locate the Keylime registrar and verifier to `driver_info.`

During the image building process the node image will be set up with an instance of the Keylime agent, as well as TPM, and IMA configurations which will allow the Keylime agent to run. The Keylime agent will register itself with the Keylime registrar automatically once started. At this point booting has begun and the node may send its first heartbeat back to the Ironic controller.

Next, start_attestation() will be called to send the allowlist and excludelist to the verifier. The conductor will make an rpc call to the agent to retrieve the Keylime agents uuid, the Keylime agents address, and the port which the Keylime agent is listening on. The Ironic controller will save these variables as `driver_info\keylime_agent_uuid`, `driver_info\keylime_agent_address`, and `driver_info\keylime_agent_port` for further use. If the conductor does not receive these credentials cleaning will fail.

The allowlist and excludelist will be sent to the verifier by calling the keylime_tenant cli programmatically. Once the verifier has received the allowlist and excludelist, attestation will begin. The verifier will periodically poll the Keylime agent for IMA measurements and compare them with the allowlist and excludelist to determine if the node has been tampered with. The verifier will record the status of the node, but take no action on the status.

At this point, the conductor may perform a validate_security_status() call to get the status of the node. If the status is what we expect, we may proceed. If the status is something we do not expect, or the controller is unable to access the verifier due to network issues, we will fail the deployment.

The Keylime agent will need to be unregistered with a call to unregister_node() to instruct the Keylime verifier to end its connection and remove the node from its database.

Here is a diagram for the anticipated workflow:

diagram { Image; Node; Keylime-tenant; Keylime-verifier; Keylime-registrar; activation = none; span_height = 1; edge_length = 250; default_note_color = white; default_fontsize = 12; Image -> Node [label = The node is booted with an image generated by diskimage-builder tool. Keylime and TPM environment is setup in the image]; Node -> Keylime-registrar [label = Makes a post request to register the Keylime agent on the node]; Keylime-registrar -> Node [label = Responses the node with an encrypted AIK]; Node -> Keylime-registrar [label = Makes an activation request with an ephemeral registrar key from TPM]; Keylime-registrar -> Node[label = 200 OK]; Node -> Keylime-tenant [label = First heartbeat]; Keylime-tenant -> Keylime-tenant [label = The allowlist and excludelist are provided by the user to the Keylime tenant command]; Keylime-tenant -> Keylime-verifier [label = Sends allowlist and excludelist and adds the Keylime agent uuid to the verifier]; Keylime-tenant -> Node [label =Gets TPM quote from the node to check the Keylime agents validity with the registrar]; Keylime-verifier -> Node [label =Starts polling the node for verification]; Keylime-tenant -> Keylime-verifier [label = Gets the current status of the node]; }

Workflows which allow node lessees to bring their own Keylime instance in to attest a node is theoretically possible within the framework given in this spec. However, Keylime currently lacks certain features needed to make this fully automated in Ironic.

Nova driver impact

None

Ramdisk impact

To have the Keylime agent work with TPM 2.0, certain libraries and configuration must be provided. These enhancements will come as part of the ramdisk. This includes tpm2-tss software stack, tpm2-tools utilities, and, although not required, the tpm2-abrmd resource manager.

Keylime-agent will be setup on the ramdisk. A new dib element will be created to install keylime-agent and make it run as a system service.

A new IPA extension will be needed to collect and send back to the conductor the keylime_agent_uuid, keylime_agent_address, and keylime_agent_port.

Security impact

It has a positive impact on security, since we can verify if the node is trustworthy by the attestation service.

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

The `attestation` interface will not be enabled by default, since the default will map to a `no-attestation` interface.

Config options

Options for configuring whether or not cleaning and deployment should fail in face of attestation failure will be part of the new `[keylime]` section

fail_clean_on_attestation_failure
: Boolean to determine whether to fail clean on attestation failure

fail_deploy_on_attestation_failure
: Boolean to determine whether to fail deploy on attestation failure

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
: Leo McGann <ljmcgann> lmcgann@redhat.com Danni Shi <sdanni> sdanni@redhat.com

Other contributors:
: None

Work Items

- Add `attestation_interface` database field.
- Implement base interface addition
- Implement `no-attestation` interface.
- Add node RPC object field
- Add API support and microversion.
- Implement Keylime attestation interface.

Dependencies

None

Testing

Testing for this interface and basic functionality, as well as integration testing using the ansible-keylime-tpm-emulator for TPM emulation.

Upgrades and Backwards Compatibility

No issues are anticipated.

Documentation Impact

Documentation will be provided about how to use keylime-verifier and keylime-registrar.

References

https://github.com/keylime

5.15.112 Provide Redfish BIOS Attribute Registry data in API

https://storyboard.openstack.org/#!/story/2008571

This proposal is to retrieve the Redfish BIOS Attribute Registry when using the Redfish driver and provide it as additional data in the bios API endpoint.

Problem description

The Redfish BIOS Attribute Registry (aka BIOS Registry) is defined in the DMTF Redfish specification[0] as a way to describe the semantics of each property in a BIOS settings resource. This JSON encoded registry contains descriptions of each property and other information, such as data type, allowable values, and whether the attribute is read-only.

When a user attempts to set a BIOS attribute through clean steps[1] it is not obvious which name should be used for the attribute, what its allowable values are, or even if the attribute is writeable. It is useful to expose the BIOS Registry through an API so that it can be used as a form of documentation when setting BIOS attributes.

An example of an entry in BIOS Registry returned from a BMC for an attribute:

```
{
  "AttributeName": "SriovGlobalEnable",
  "CurrentValue": null,
  "DisplayName": "SR-IOV Global Enable",
  "DisplayOrder": 2331,
  "HelpText": "Enables or disables the BIOS configuration of Single Root
              I/O Virtualization (SR-IOV) devices. Enable this feature
              if booting to a virtualization operating system that
              recognize SR-IOV devices. ",
  "Hidden": false,
  "Immutable": false,
  "MenuPath": "./IntegratedDevicesRef",
  "ReadOnly": false,
  "ResetRequired": true,
```

(continues on next page)

[0] DMTF Redfish Specification - http://redfish.dmtf.org/schemas/DSP0266_1.11.0.html
[1] BIOS configuration - https://github.com/openstack/ironic-specs/blob/master/specs/approved/generic-bios-config.rst

(continued from previous page)

```
  "Type": "Enumeration",
  "Value": [
    {
      "ValueDisplayName": "Enabled",
      "ValueName": "Enabled"
    },
    {
      "ValueDisplayName": "Disabled",
      "ValueName": "Disabled"
    }
  ],
  "WarningText": null,
  "WriteOnly": false
}
```

This change was discussed at the Wallaby mid-cycle, see notes[2].

Proposed change

The effort will encompass the following changes:

- Support for getting the BIOS Registry in Sushy. Retrieving the BIOS Registry requires multiple Redfish requests as the base URI - `/redfish/v1/Registries`, provides the BIOS Registry URI with a vendor specific name. The registry will be a list of dictionary entries, one for each BIOS attribute.
- Changes to Ironic to retrieve the Redfish BIOS Registry for a node when it gets the BIOS settings; currently this is when Ironic moves the node to `manageable` or `cleaning` and after changing the BIOS settings.
- Filtering of the BIOS registry received from Sushy to only save the entries that match the BIOS settings received from Sushy, based on the `AttributeName` field in the BIOS Registry entries.
- Storing the registry data in the Ironic DB associated with each BIOS setting.
- Exposing the BIOS Registry data via the REST API. No new endpoints will be added. The registry data will be included in the `/v1/nodes/<node>/bios` endpoint only if `?detail=True` is set. The registry data will always be included in `/v1/nodes/<node>/bios/<setting>`. This is similar to how the API for nodes currently works e.g. `/v1/nodes`, `/v1/nodes?detail=True`, and `/v1/nodes/<node>`

Alternatives

- An operator can continue to set the BIOS settings in clean steps perhaps by using vendor documentation or only making simple changes to Boolean values.
- Instead of retrieving the registry and caching it, Ironic could do a synchronous get from the API. However, this may lead to performance problems especially when handling multiple API requests.

[2] Discussion at Wallaby mid-cycle - https://etherpad.opendev.org/p/ironic-wallaby-midcycle

Data model impact

- There is a schema defined for the BIOS Registry, for example[3], however it is vendor dependent as to which attribute fields are stored in the registry. Sushy should parse the following common fields which are present in the registry of typical vendors, for example Dell, HPE, and SuperMicro:

```
* AttributeName
* DefaultValue
* Type
* Immutable
* ReadOnly
* ResetRequired
* IsSystemUniqueProperty
* LowerBound
* UpperBound
* MinLength
* MaxLength
* Value (possible values for enumeration types)
  * ValueName
```

- The BIOS Registry will be stored in the Ironic DB along with the BIOSSetting, see[4]. The following columns will be added and must be included in the migration code:
  - `attribute_type` (string)
  - `allowable_values` (list)
  - `lower_bound` (integer)
  - `max_length` (integer)
  - `min_length` (integer)
  - `read_only` (boolean)
  - `reset_required` (boolean)
  - `unique` (boolean)
  - `upper_bound` (integer)

For each setting only the relevant fields for `attribute_type` will be stored as returned from the BMC. For example, for an Enumeration type only `allowable_values` will be stored and for an Integer only `min_length` and `max_length`.

In addition, the BIOSSetting `name` field should be indexed for faster retrieval.

State Machine Impact

The BIOS Registry will be retrieved from the BMC upon transition to `manageable` and `cleaning` states using multiple Redfish requests. This may add some additional time to the state transition but it is not foreseen that this will have an impact on the state transition performance.

[3] AttributeRegistry schema - https://redfish.dmtf.org/schemas/v1/AttributeRegistry.v1_3_4.json

[4] BIOSSetting in Ironic DB - https://opendev.org/openstack/ironic/src/branch/master/ironic/db/sqlalchemy/models.py#L326

REST API impact

- No new REST API endpoints will be added. The `/v1/nodes/<node>/bios/<attribute>` endpoint will be changed to include the new registry fields. The `/v1/nodes/<node>/bios` endpoint will only include this data per attribute if `?detail=True` is set, similar to how the `/v1/nodes/` endpoint works. The query can also be used to retrieve only particular fields.
- If BIOS Registry could not be retrieved from the node then the registry fields will be set to `null`.
- For allowable_values, some vendors include both ValueName and ValueDisplayName in the response. The API will just show a list of the allowable values.
- An example response for `/v1/nodes/<node>/bios/ProcVirtualization`, an enumeration type, with the new registry fields is as follows:


```
{ "ProcVirtualization":
    { "created_at": "2021-03-31T13:50:51+00:00",
      "updated_at": null,
      "name": "ProcVirtualization",
      "value": "Enabled",
      "allowable_values": ["Enabled","Disabled"]
      "lower_bound": null,
      "max_length": null,
      "min_length": null,
      "read_only": false,
      "reset_required": null,
      "type": "Enumeration",
      "unique": null,
      "upper_bound": null,
      "links": [
      {
        "href": "http://IP/v1/nodes/1b1c6edf-4459-4172-b069-5c6ca3e59e03/
↪bios/ProcVirtualization",
        "rel": "self"
      },
      {
        "href": "http://IP/nodes/1b1c6edf-4459-4172-b069-5c6ca3e59e03/
↪bios/ProcVirtualization",
        "rel": "bookmark"
      }
    ]
  }
}
```


- An example response for `/v1/nodes/<node>/bios/SerialNumber`, a String type unique to this node, is as follows:


```
{ "SerialNumber":
    { "created_at": "2021-03-31T13:50:51+00:00",
      "updated_at": null,
      "name": "SerialNumber",
      "value": "Q95102Q8",
      "allowable_values": [],
```


(continues on next page)

(continued from previous page)

```
      "lower_bound": null,
      "max_length": 16,
      "min_length": null,
      "read_only": false,
      "reset_required": null,
      "type": "String",
      "unique": true,
      "upper_bound": null,
      "links": [
      {
        "href": "http://IP/v1/nodes/1b1c6edf-4459-4172-b069-5c6ca3e59e03/
↪bios/SerialNumber",
        "rel": "self"
      },
      {
        "href": "http://IP/nodes/1b1c6edf-4459-4172-b069-5c6ca3e59e03/
↪bios/SerialNumber",
        "rel": "bookmark"
      }
    ]
  }
}
```

Client (CLI) impact

baremetal CLI

- The bios command `baremetal node bios` will change as follows.
  - This command will now include the registry data as an additional fields, for example:

```
$ baremetal node bios setting show hp-910 ProcVirtualization -f json
{
  "created_at": "2021-03-31T13:50:51+00:00",
  "name": "ProcVirtualization",
  "updated_at": null,
  "value": "Enabled"
  "allowable_values": ["Enabled","Disabled"]
  "lower_bound": null,
  "max_length": null,
  "min_length": null,
  "read_only": false,
  "reset_required": null,
  "type": "Enumeration",
  "unique": null,
  "upper_bound": null,
}
```

- A new parameter `--long` will be added to the `baremetal node bios list` command to include the registry data for each attribute. It will not be included by default. This is similar to `baremetal node list <node> --long` command.

openstacksdk

Openstacksdk does not currently have support for bios settings. Although not a requirement for this specification. bios should be added to openstacksdk and the detailed registry information should be included.

RPC API impact

A new method to get the BIOS registry will be added to the RPC API.

Driver API impact

- The `cache_bios_settings` method in the Redfish driver will now also get the BIOS Registry from Sushy.

Nova driver impact

None

Ramdisk impact

None

Security impact

Unprivileged access to the BIOS configuration can expose sensitive BIOS information and configurable BIOS options to attackers, which may lead to disruptive consequence. Its recommended that this kind of ability is restricted to administrative roles.

Other end user impact

None

Scalability impact

The Redfish requests to retrieve BIOS Registry data will increase the traffic to the BMC however, since these requests will only be when the node transitions to `manageable` or `cleaning`, it should not impact scalability.

The BIOS Registry data will be stored in the Ironic DB along with the BIOS setting that it corresponds to. This will add to the size of the Ironic DB but its not expected to be prohibitive as the BIOS settings list, although it varies per vendor, is approximately 100 to 200 items.

Performance Impact

It is not expected that this change will introduce a performance impact on the time it takes for nodes to transition to `manageable` or `cleaning`.

Other deployer impact

None

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
bfournie@redhat.com

Work Items

- Add support for BIOS Registry to Sushy
- Add caching support for the registry to Ironic.
- Add retrieval of the registry in Ironic when transitioning the node to `cleaning` or `manageable` and store/update the cache.
- Add the API to get the BIOS Registry for the node.
- Add configuration items for `bios_registory_lang` and `bios_registry_enable` to ironic.conf.

Dependencies

None

Testing

- Unit testing will be added with sample vendor data but it will necessary for 3rd party testing to be added to test each vendors interface.

Upgrades and Backwards Compatibility

None

Documentation Impact

- API reference will be updated.

References

5.15.113 Boot configuration API

https://bugs.launchpad.net/ironic/+bug/2044561

Problem description

Currently, an Ironic conductor generates various boot files (e.g. iPXE scripts) in the public directory of the HTTP server associated with it. Then dynamic DHCP configuration with Neutron is used to point a node at the right HTTP server.

For standalone installations, the DHCP configuration is usually static. It works well with a single conductor, but is not operational out-of-box in a multi-conductor setup: a node simply does not know which HTTP server to boot from.

Proposed change

New API will be added to Ironic to serve boot configuration files, with iPXE being the reference implementation. This approach will allow us to rely on the existing hash ring to direct the request to the right conductor. See *REST API impact* for further details.

This is how the boot process will work in the case of iPXE, standalone Ironic and static DHCP configuration:

1. The Nodes iPXE firmware initiates a new DHCP session.
2. The DHCP request reaches the Ironics DHCP server (usually dnsmasq).
3. The DHCP server responds with an IP address and the catch-all boot script. This boot script will be located next to an arbitrary conductor, quite likely the one co-existing with the DHCP server.
4. The boot script will direct the iPXE firmware to the URL in the form of `http://<IP>:6385/v1/boot/<MAC>/boot.ipxe` where `<IP>` is the IP of any Ironic API instance (most likely the one co-existing with the HTTP server), `<MAC>` is the nodes MAC address.
5. Ironic API will find the node by MAC and direct the request to the correct conductor.
6. The conductor responsible for the node returns the iPXE script.

Alternatives

One proposed approach is to enable coordination between conductors. While potentially useful in more cases, that proposal has a JSON RPC multiplication problem in a standalone setup.

The other approach to the problem is to introduce managed standalone DHCP. The first step towards it has already been done: Ironic can manage its dnsmasq server. In a multi-conductor setup, it may imply having several dnsmasq instances on the same network, which is a potentially problematic setup. The current implementation still requires some sort of coordination between conductor or a periodic task to disable access to the DHCP servers of unrelated conductors.

A much better, although also the most complex, option would be to use an existing DHCP server with API access. One such server is Kea. The main problem: such a complex change may be too much for Ironic right now. The contributors are spread thin already. On top of that, Ive been told that only paid addons give us what we need.

Data model impact

None

State Machine Impact

None

REST API impact

`GET /v1/boot/<MAC>/<NAME>`

<MAC>
MAC address of any NIC of a node.

<NAME>
Configuration name to request (e.g. `boot.ipxe`).

The API will find the node by the MAC address, check its provision state and call into the `get_boot_config` RPC on success. On failure, a generic HTTP 404 will be returned to avoid disclosing any further information.

> **Note**
>
> The whole API branch `/v1/boot` will not be versioned since we dont expect firmware implementations to support extra headers or any sort of reasonable version negotiation.

Client (CLI) impact

None. The API is not for end users.

openstack baremetal CLI

None

openstacksdk

None

RPC API impact

```
def get_boot_config(self, context, node_id, name):
    """Request a boot configuration file."""
```

Driver API impact

The `boot` interface will get a new call:

```
def get_boot_config(self, task, name):
    """Request a boot configuration file.

    :param task: a TaskManage instance with a shared lock.
    :param name: configuration name.
    :raises: NotFound if the configuration cannot be produced.
    """
```

(continues on next page)

(continued from previous page)

```
    raise exception.UnsupportedDriverExtension(
        driver=task.node.driver, extension='get_boot_config')
```

A reference implementation will be added to the iPXE boot interface, supporting a configuration called `boot.ipxe` - the iPXE script.

Nova driver impact

None

Ramdisk impact

None

Security impact

The new API will allow enumerating nodes in certain states by their MAC addresses. Some information may potentially be exposed by the boot configuration. The enumeration is already possible with the lookup API, and the configuration can be leaked by the boot scripts in the HTTP server. We will advise operators to limit access to the new API endpoints.

On top of that, the boot interfaces `validate` will not be called to avoid exposing information about the node fields.

Other end user impact

None

Scalability impact

Serving boot scripts via the API is somewhat less efficient than from an HTTP server. Operators concerned about the impact can opt into using the old approach.

We will avoid using an exclusive lock or launching additional threads in the implementation. The initial version will just read the existing files from disk.

Performance Impact

None

Other deployer impact

The feature will be configured with a few new options:

`[pxe]ipxe_use_boot_config_api = False`
: Enables the feature. If true, the generated root iPXE script (`boot.ipxe`) will contain links to the boot configuration API, not to the HTTP server.

`[pxe]ipxe_config_api_root_url = <None>`
: Specifies the root URL to use for links to the boot configuration API. An example use case is an Ironic deployment with TLS: iPXE does not support custom certificates without recompiling the firmware, so e.g. a proxy must be established instead.

`[api]restrict_boot_config = True`
: Instructs the API to be restricted to only nodes in * `WAIT` states. Operators using fast-track may want to set this to False.

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
Dmitry Tantsur (dtantsur)

Work Items

TODO

Dependencies

None

Testing

We can switch Bifrost to the new API. Its highly likely that Metal3 will also switch to it in the near future as we develop its HA story.

Upgrades and Backwards Compatibility

None.

Documentation Impact

Installation guide may need to be adjusted.

References

5.15.114 Boot from a Ramdisk

https://storyboard.openstack.org/#!/story/1753842

While Ironic, by its very operational nature supports booting from a ramdisk, we do not empower or enable operators who wish to utilize instances that boot from ramdisks.

In the use cases desired, these are often ephemeral instances, with no actual back-end storage. That being said, we of course cant forget cleaning and other mechanisms that help provide improved value for Ironic in a deployment.

Most of these cases are where an operator is sufficiently knowledgeable to construct a ramdisk and kernel image that can be explicitly booted from RAM. Such cases are common in High Performance Computing environments where instances may need to be quickly deployed, or are functionally disk-less in order to remove a point of failure.

Problem description

- Operators desire fully ephemeral instances that essentially operate in RAM only. This allows compute-local disks to be dedicated to ephemeral storage, or to be omitted from the machine entirely.
- Large scale cluster computing practices have long used prepared kernel/ramdisk images in order to help facilitate the stand-up and operation of their environments.
- Ironic has _some_ of the support required. We do something somewhat similar with boot from volume, and if we ever support RBD booting, we would functionally require this exact feature. What delineates this from the Boot from Volume iPXE chain loading scenarios and virtual media based booting from a remote volume, is that Boot from Volume scenarios are ultimately reliant upon back-end block storage, which is not always needed or desired for all cases.

Proposed change

We propose to build a `ramdisk` deploy interface built upon our existing ramdisk code and facilities. This would allow us to have a jump start on working functionality and help us better identify where refactoring must be required.

The `ramdisk` deployment driver interface would be an opt-in interface which focuses on supporting ramdisk booting cases.

The `ramdisk` deployment driver would also loop in the agent interface classes, in order to minimize complexity and allow for cleaning workflow and related code to remain in one location.

Mechanism wise, if the `ramdisk` deployment interface is set:

- The instance can boot based upon any provided instance `kernel` and `ramdisk` as part of the requested glance image to be deployed. Effectively any image contents would be ignored.
- The instance `boot_option` will be ignored by the ramdisk interface, and will be explicitly set to `ramdisk`. If otherwise set, the interface logs a warning.
- The same `kernel` and `ramdisk` parameters that are used for a glance image based deployment could be re-used to contain a URL.
  - In the future, ironic should consider extending the accepted URL format to include `nfs://${server_ip}{$path}{$file}`, which would enable direct boot to NFS hosted kernel and ramdisk. This would not consist of `nfsroot` scenarios. This document does not require that is performed, but the Boot from Volume specification does explicitly state that could be a scenario implemented.
  - An additional consideration could also be to support direct specification of a path to chainload to.
- Users must explicitly allow TFTP and/or HTTP iPXE endpoint access for booting the baremetal nodes. Typically deployments do not allow tenant networks to access to these endpoints.
- Configuration drives would be ignored by ironic, and users would be advised to make use of the metadata service. Deployments with a configuration drive will explicitly result in an warning being logged by the conductor.
  - Operators who directly deploy nodes using ironic may need to pass additional kernel command line arguments to the node being deployed via a ramdisk. In this case, a `/instance_info/ramdisk_kernel_arguments` field will be accepted to allow those operators to pass information to the instance.

Building this functionality would allow us to also quickly adapt the support to enable Ceph RBD based booting as well as ramdisk booting to an NFS share serving as the root device. Both are features some operators have expressed desire for. Ceph RBD support would need to be in the underlying `pxe` boot interface, as the ramdisk interface overlays the `pxe` interface. Support for Ceph RBD or NFS based booting, from `volume target` information, would be a future enhancement.

Alternatives

We could consider building logic to handle and support this into the standard deployment workflow, however given this is a sufficient and specialized use case, that it might be completely unnecessary as it would not be used in most cases.

Data model impact

None.

State Machine Impact

None. Deployment would consist of setting the boot configuration and then powering on the network node.

REST API impact

None

Client (CLI) impact

ironic CLI

None

openstack baremetal CLI

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

None

Security impact

No additional security impact is expected aside from a deployment scenario being able to be setup by an operator where it may be necessary to keep the deployed kernel/ramdisk potentially for an elongated period of time.

Ironic already does this with instances that are netboot.

Other end user impact

Operators which choose to use this feature may wish to put in place specialized network controls to facilitate the machines network booting.

Each deployment and case will be different, and without post-implementation information, we will be unable to determine if there is a standard that can be derived.

Scalability impact

No scalability impact anticipated.

Performance Impact

No substantial performance impact anticipated, although if the feature gains popularity takeover naturally takes longer.

Other deployer impact

None

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
Julia Kreger (TheJulia)

Other contributors:
None

Work Items

- Create deploy interface
- Create tempest test for said ramdisk deploy interface.
- Create user documentation.

Dependencies

Testing

This seems like a feature that could be easily tested via a tempest scenario if the driver is available. No additional testing should be required.

Upgrades and Backwards Compatibility

None

Documentation Impact

Documentation will need to be updated to support this effort.

References

None

5.15.115 Boot from Volume - Reference Drivers

https://bugs.launchpad.net/ironic/+bug/1559691

In order to support booting ironic nodes from a storage device that is hosted and/or controlled remotely, such as via a SAN or cinder iSCSI target, changes are required to ironic in order to help facilitate and enable drivers to leverage the remote storage as a boot volume.

Problem description

- Presently ironic has no support in the reference drivers to boot a node from a remote volume.
- Due to the lack of this support, users capabilities and usability are limited. Providing this functionality allows deployments to leverage greater capabilities.

Boot from Volume scenarios

This specification is broken into multiple portions in order to facilitate a phased implementation of support. While it would appear that these changes should be limited to the boot and deployment interfaces, they ultimately require substantial substrate changes to provide necessary functionality.

As there is a large variety of potential connection methods and configurations, the scenarios we have identified appear suitable for implementation in a reference driver. We are primarily focused on the use of cinder, but intend to make the storage substrate pluggable and generic.

Not covered as part of this specification is handling of support for MultiPath IO, sometimes referred to as MPIO. We feel this is largely outside of Ironics scope at this point in time.

Additionally not covered is the requirement to use UEFI HTTP boot support, which was set forth during the ironic mitaka midcycle. Using UEFI HTTP boot would simply load the iPXE UEFI boot loader, and would thus be a redundant capability compared with existing capabilities.

Below are various boot from volume scenarios. These scenarios are intended to provide a clear definition of conditions and requirements.

Scenario 1 - iPXE boot from iSCSI volume

Conditions:

- Node boot defaults to network booting via iPXE controlled by the ironic conductor. This is similar to the PXE driver netboot option.
- Metadata service use is expected as configuration drive support is not feasible at this time.
- **Configuration drive is not supported.**
  - For context: This is a limitation as a result of the minimum new volume and new volume extension sizes in cinder. This may be a capability at a later point in time, but is out of scope for an initial implementation.
- iPXE version sufficient to support boot from iSCSI target.
- Operating system is already deployed to the intended iSCSI boot volume via external means.
- **Tenant network isolation is unsupported due to the need for iPXE boot.**
  - Potentially in the future, a framework or proxy could be created to enable tenant network isolation, however that is considered out of scope at this time.
- Node is configured by the operator with the `node.properties['capabilities']` setting of `iscsi_boot` set to a value of `true` for node validation to succeed.
- Node has a defined volume target with a `boot_index` value of `0` in the volume_targets table.

Requirements:

- **Storage driver interface**
  - A storage driver/provider interface is needed in order to support recovery of systems in the event of a failure or change in hardware configuration.
- iPXE template and generation logic changes.
- Implementation of substrate logic in the deploy and PXE driver modules to appropriately handle booting a node in a boot from volume scenario.

Scenario 2 - iPXE boot from Fibre Channel over Ethernet (FCoE)

Conditions:

- Node boot defaults to network booting via iPXE controlled by the ironic conductor. Similar to the PXE driver netboot option.
- Metadata service use is expected.
- Configuration drive is not supported.
- Operating system is already deployed to the intended boot volume via external means.
- Tenant network isolation is unsupported due to iPXE requirements.

- Node is configured with the `node.properties['capabilities']` setting of `fiberchannel_boot` set to a value of `true` for node validation to succeed.
- Node has a defined volume target with a `boot_index` value of `0` in the volume_targets table.

Requirements:

- *Scenario 1 - iPXE boot from iSCSI volume* implemented
- Additional logic for iPXE template generation in the FCoE use case

Scenario 3 - Boot via hardware Host Bus Adapter (HBA) to volume

Scenario involves boot to a local volume, however, the volume is expected to be pre-populated with a bootable operating system. No volume deployment operation is expected.

This scenario is based upon the expectation that the environment and the nodes firmware settings are such that it is capable of booting directly to a presented volume via the HBA once powered on.

Conditions:

- Metadata service use is expected.
- Configuration drive support is unavailable.
- Node boot defaults to block storage device.
- Tenant network isolation is supported as this scenario does not require iPXE booting for normal operation of the host.
- Operating system is already deployed to the intended boot volume via external means.
- Infrastructure and HBA BIOS configuration is such that the node will boot to what is expected to be the Logical Unit Number (LUN) offered as the boot volume.
- Node is configured with the `node.properties['capabilities']` setting of `hba_boot` set to a value of `true` for node validation to succeed.
- Node has a defined volume target with a `boot_index` value of `0`.

Requirements:

- Creation of driver deploy method functionality that under normal circumstances performs a no-op, and defaults the next boot state to local disk when the user requests a node to be deployed.

Scenario 4 - Boot via Host Bus Adapter (HBA) to volume with image deployment

Scenario involves boot to a local volume where the volume needs to be written out by Ironics mechanisms.

Conditions:

- Metadata service is not required.
- Configuration drive is supportable.
- Node boots to block storage device.

- Tenant network isolation is supported as this scenario does not require iPXE booting for normal operation of the host.
- Infrastructure and HBA BIOS configuration is such that the node will boot to what is expected to be the LUN offered as the boot volume.
- Node is configured with the `node.properties['capabilities']` setting of `hba_boot` set to a value of `true`.
- Node is configured with the `node.instance_info['force_deployment']` parameter set to ``true`.

Requirements:

- This method is expected to be essentially identical to the scenario defined in *Scenario 3 - Boot via hardware Host Bus Adapter (HBA) to volume*, however via the inclusion of default logic that only invokes deploy phase when explicitly requested for boot from a volume.

Scenario 5 - iPXE boot to NFS root volume

Scenario involves use of a kernel and ramdisk image hosted by the conductor host in order to enable the node to boot via iPXE with sufficient command line parameters to enable the root volume to attach during the boot sequence.

This is a logical progression given that users have indicated that they have enabled similar boot functionality downstream.

Conditions:

- Metadata service use expected.
- Configuration drive support is unavailable.
- Node boot defaults to iPXE.
- Node boot utilizes kernel and ramdisk hosted by the conductor.
- Operating System is already deployed to the intended boot volume via external means.
- Tenant network isolation is unsupported due to iPXE need.
- Node is configured with `node.properties['capabilities']` setting of `nfs_boot` set to a value of true coupled with an `instance_info` setting of `nfs_root` which provides nfs root connection information.
- Kernel and ramdisk utilized support the nfsroot kernel command line option.

Potential future capabilities

These are some additional items that it might make sense to develop later on after reference driver implementation has been completed, however these are out of scope of the existing specification.

- Boot the agent from a remote volume to facilitate a deployment.
- Creation of a deployment framework to allow IPA to potentially apply HBA settings.
- Multipath IO configuration handling and potentially passing.

- Configuration drives support.
- Support for tenant network isolation boot scenarios.

Proposed change

In order to support an initial feature set, i.e. Scenarios 1-4, for boot from volume, we propose the following:

- Implementation of a basic capability concept via a helper method that will allow a driver/provider capability to be checked by another portion of the code base, returning false if the capabilities definition is missing. Example:

```
utils.check_capability(
    task.driver.boot.capabilities, 'iscsi_volume_boot')
utils.check_compatibility(
    task.driver.deploy.capabilities, 'iscsi_volume_deploy')
```

  This would be implemented via a list of capability tags to each main level interface class, as required in order to guard against invalid configurations.

- Implementation of logic in the existing reference driver deploy validate methods to fail the validation of any node that has volume storage configured when the driver that a node is configured with lacks any such support as identified by the previously noted capability interface. This is to help ensure that such nodes are not accidentally provisioned with erroneous user expectations.

- Updating of the `agent.AgentDeploy` and `iscsi_deploy.ISCSIDeploy` driver logic to support skipping deployment of a node if a storage interface provider is defined, volume information exists for the volume, and the volume has an index of `0` indicating it is the boot volume. In essence, this means that if the node is defined to boot from a remote volume, that the driver.deploy.deploy method should immediately return DEPLOYDONE as any network booting configuration, if applicable, would have to be written out via driver.deploy.prepare method. Example:

```
if (task.node.storage_interface is not None and
        not task.driver.storage.should_write_image(task)):
```

  Additionally, validation logic will need to be updated in the deploy drivers to pass specific checks of instance_info fields that do not apply with the case of booting from a volume.

- **Creation of a storage provider interface:**

  - Similar in composition to the network provider interface, where a default will result in a provider interface that performs `no-op` behavior, while exposing an empty set of storage capabilities.
  - A node level `storage_interface` setting with default value, similar to the Add network interface to base driver class change, to define if a node is to utilize the storage interface along with the provider that is to be utilized. This is intended to align with the Driver composition reform specification.
  - Initial and reference storage driver to be based upon cinder, leveraging `python-cinderclient` for the REST API interface, to support the following fundamental operations detailed below.
    - `detach_volumes` - Initially implemented to enumerate through known attached volumes and remove the storage attachments.

    * `attach_volumes` - Initially implemented to enumerate through known configured volumes and issue storage attachment requests. In the case of the cinder driver, we will reserve the volumes in our configuration, initialize connections to the volumes meanwhile supplying current initiator configuration, then trigger the cinder volume attach call to update the database configuration. Additionally, we will update the volume metadata to allow for easy user identification of volumes that are used for ironic nodes by recording information to allow for reconciliation of nodes that are powered off with detached volume.
    * `validate` - Initially implemented to validate that sufficient configuration is present in the configuration to allow for the functionality to operate. This validation will be skipped if no volumes are defined.
    * `should_write_image` - Provides an interface to allow the other components to ask the storage driver if the image should be written to disk. This allows all of the logic to be housed with the storage interface.

- Updating of the `pxe.PXEBoot` driver logic to support the creation of the appropriate iPXE configurations for booting from the various boot scenarios if the `volume_target` information is defined, iPXE is enabled, and a storage provider is available.
- Updating of the `pxe.PXEBoot` validate interface to leverage a helper method when a storage provider and boot volume is defined in the node configuration, to validate that the capabilities, initiator configuration, and volume configuration are in alignment for the specified volume/path type. Note that the bulk of this logic should reside in a `deploy_utils` method that can be re-used by other drivers moving forward.
- Updating of the conductor `utils.node_power_action` logic to enable the storage provider (defined by the node.storage_interface setting) to be called to permit volume attachment and detachment for the node and thus update any storage configuration if necessary.
- Addition of a helper method that sets and returns, if not already set, the node.instance_infos boot_option parameter based upon the hardware configuration, and supplied volume configuration information, enabling matching and identification of what the next appropriate step is for each scenario.
- Updating of the conductor `_do_node_tear_down` method to call the storage provider detach_volumes method and purge volume information if not already implemented.
    - At the beginning and completion of the storage detachment interaction, a notification shall be generated to allow visibility into if the process is successfully completed.
- Updating the iPXE template logic to support the creation of the various file formats required for Scenarios 1, 2, 5. See: IPXE sanhook command, IPXE san connection URLs and nfsroot kernel command line option.

As previously noted, each scenario will be submitted separately as incremental additions to ironic.

In order to support scenario 4, the deploy driver will need to understand how to deploy to a system with such configuration.

- Updating of the deploy driver to enable the logic added for scenarios 1-3 to be bypassed using a `force_deployment` parameter which should be removed from the nodes instance_info prior to the node reaching the active state. This, in effect, would cause ironic to support a deployment operation when the supplied volume information is normally expected to have a valid Operating System and boot loader already in place.

- Agent will need to be informed of the desired volume for the boot volume, and, if supplied to the target, connection information. The appropriate information should be passed in using Root device hints, specifically setting a WWN or volume serial number if available.

In order to support scenario 5:

- Scenario 3 must be implemented. This is anticipated to largely be an alternative path in the iPXE template where the previously defined settings cause the on-disk PXE booting template to boot the node from the NFS root.

Later potential improvements above and beyond this initial specification:

- Creation of logic to allow Ironic users to leverage the storage provider to request a volume for their node. Such functionality would require ironic to deploy the OS image, and should be covered by a separate specification.

Alternatives

An alternative could be to simply not develop this functionality and to encourage downstream consumers to independently develop similar tooling to meet their specific environments needs. That being said, both options are unappealing from the standpoint of wishing to grow and enhance ironic.

Data model impact

A *storage_interface* field will be added to the node object which will be mapped to the appropriate storage driver.

State Machine Impact

None

REST API impact

As the node storage driver will be selectable by the operator, it will need to be concealed from older API clients, which will necessitate a microversion update once the functionality is present in the API.

Client (CLI) impact

ironic CLI

None. All expected CLI updates are expected to be part of the specification covering information storage, Add volume connection information for Ironic nodes.

openstack baremetal CLI

None

RPC API impact

None

Driver API impact

The first change is the introduction of a list of supported advanced driver features defined by the deploy and boot driver classes, known as capabilities, that allow for other driver components to become aware of what functionality is present in a neighboring driver interface/composition.

The second change is the introduction of a new *storage_interface* that will be mapped to a selectable storage driver of the available storage drivers.

Within this storage interface, new methods will be defined to support expected storage operations. Below are the methods that are anticipated to be added and publicly available from drivers:

```
def attach_volumes(self, task):
"""Informs the storage subsystem to detach all volumes for the node."""

def detach_volumes(self, task):
"""Informs the storage subsystem to detach all volumes for the node."""

def validate(self, task):
"""Validate that the storage driver has appropriate configuration."""

def should_write_image(self, task):
"""Determines if deploy should perform the image write-out."""
```

Nova driver impact

Impact to the nova driver is covered in a separate specification Nova Specs - Ironic - Boot from Cinder volume. As the driver will be available as an opt-in change, we expect no negative impact on behavior.

Ramdisk impact

While we should be able to derive the intended root volume and pass an appropriate root hint if necessary in order to facilitate a deployment as part of *Scenario 4 - Boot via Host Bus Adapter (HBA) to volume with image deployment* , the IPA ramdisk should likely have functionality added in the form of a HardwareManager in order to support MutliPath IO. That being said MPIO is out of scope for this specification.

Security impact

Overall, the storage driver, in this case, cinder, will need to utilize credentials that are already populated in the configuration for keystone to connect to cinder to obtain and update mapping information for volumes.

Scenarios 1-2 and 5 are designed such that the tenant machine is able to reach the various services being offered up by whatever the volume driver is set to leverage over the nodes default network configuration. As a result of the need to network boot, the flat network topology is required along with access controls such that the nodes are able to reach the services storage volumes.

The more secure alternative are the drivers representing scenarios 3 and 4 as this configuration ultimately requires a separate storage infrastructure. This case will allow for tenant network isolation of deployed nodes.

Other end user impact

This functionality may require additional knowledge to be conveyed to the ironic-webclient and ironic-ui sub-projects, however that will need to be assessed at the time of implementation as they are under active development.

Scalability impact

This is a feature that would be opted into use by an operator. In the case where it is active, additional calls to the backend storage interface may have a performance impact depending upon architecture of the deployment.

Performance Impact

For each node under ironics care that we believe has volumes, we need to query storage manager, presumably cinder based on this implementation, and attach/detach volumes during intentional user drive power operations. This may extend the call to power-on a node after deployment, or potentially prevent power-up if the attachment cannot be made.

Other deployer impact

Deployers wishing to use these drivers will naturally need to add the `cinder` storage interface to the `enable_storage_interfaces` list.

A default storage configuration driver will be set to `noop` which will prevent any storage related code from being invoked until the operator explicitly chooses to enable this support.

Based on the proposed driver configuration, we can expect two additional sections in the conductor configuration file:

```
[DEFAULT]
enabled_storage_interfaces = <None> # Defaults to none and is a list of the
available drivers.

[cinder]
url = http://api.address:port
url_timeout = 30
retries = 3
auth_strategy = <noauth|keystone>
toggle_attachments_with_power = true
```

Developer impact

This will increase the overall complexity and set of capabilities that ironic offers. Driver developers wishing to implement driver specific functionality should expect certain substrate operations to occur, and attempt to leverage the substrate set forth in this specification and the Add volume connection information for Ironic nodes specification.

Implementation

Assignee(s)

Primary assignee:
juliaashleykreger

Other contributors:
blakec shiina-hironori

Work Items

The breakdown of the proposed changes, when combined with the underlying scenarios helps convey the varying work items. That being said, this functionality will take some time to land.

Dependencies

Logic will need to be implemented in IPA to handle the scenario when no disks are detected. `cleaning` and `inspection` operations should be able to be executed upon hardware nodes that have no local disk storage.

Additionally in IPA, as a soft dependency, logic MAY be required to better handle directly attached volume selection when multipathing is present. This will require its own specification or well-defined and validated plan as IPA cannot expect OS multipathing support to handle MPIO scenarios in all cases, or to even be present.

Implementation of Add volume connection information for Ironic nodes. This specification should not be entirely dependent upon the implementation of the Nova Specs - Ironic - Boot from Cinder volume specification.

A soft dependency exists for the Driver composition reform specification in that these two efforts are anticipated to be worked in parallel, and this implementation effort should appropriately incorporate functionality as the functionality for the Driver composition reform specification begins to become available.

Testing

The level of full stack functional and integration tests is a topic that requires further discussion in the ironic community. An initial case for a gate test could be where an ironic deployment boots from a Cinder volume, which a tempest test could orchestrate.

Scenarios 3 and 4 are the most difficult to test as they have detached infrastructure expectations outside of our direct control. However, we may find that the base overlay is sufficient to test with unit tests due to what will ultimately be significant underlying common paths.

Upgrades and Backwards Compatibility

As this feature set is being created as a new set of capabilities within the reference drivers and their capability, no compatibility issues are expected as the API field additions related to this specification will be hidden from an API client that does not request the appropriate API version.

A database migration step will be added to create the `storage_interface` node database field. The initial value for this field will be None, and there will be no implied default set as an operator must choose to enable a storage interface in their environment.

Documentation Impact

Documentation will need to be updated detailing the new driver and the related use scenarios so an operator can comprehend what options the driver(s) provide and how they can fit into their use cases. Additional caveats regarding long term network booting of hosts should be explicitly stated as part of this work.

It is expected that this specification will be further refined during development of this functionality in order to raise and document any new findings at a technical level.

References

Relevant specifications:

- Add volume connection information for Ironic nodes
- Nova Specs - Ironic - Boot from Cinder volume

Mitaka midcycle etherpad:

- ironic mitaka midcycle

5.15.116 Discover node properties and capabilities for ucs drivers

https://bugs.launchpad.net/ironic/+bug/1526359

This proposal adds the ability to inspect/update hardware properties and auto-create ports for Cisco UCS B/C/M-servers managed by Cisco UCS Manager. It uses out-of-band H/W inspection utility provided by UcsSdk.

Note

This specification has been retired as the vendor specific UCS driers are no longer available in Ironic. This was a result of the Open Source UcsSdk library no longer being maintained.

Problem description

Node inspection automatically collects node properties. These properties are required for scheduling or for deploy (ports creation). Today UCS drivers doesnt support node inspection. Enhance UCS drivers to support node inspection.

Proposed change

This spec proposes change to enhance UCS drivers to support node inspection that discovers node properties and capabilities of Cisco UCS B/C/M-series servers managed by Cisco UCSM. This is done by using out-of-band H/W inspection interface provided by UcsSdk Python library.

The following mandatory properties will be discovered and updated in node.properties as discussed in http://specs.openstack.org/openstack/ironic-specs/specs/kilo/ironic-node-properties-discovery.html

- memory size
- CPUs
- CPU architecture
- NIC(s) MAC address

- disks

The following additional properties are of interest to UCS drivers and will be discovered and updated to node.properties as capabilities:

- UCS Host Firmware pack

  capability name : ucs_host_firmware_package possible values : it can vary hardware to hardware.

- Server Name/Model

  capability name : server_model possible values : it can vary hardware to hardware.

- RAID level

  capability name : max_raid_level possible values : 0,1,5,6,10

- secure boot capability

  capability name : secure_boot possible values : true, false

- PCI (GPU) devices

  capability name : pci_gpu_devices possible values : count of such devices.

- SR-IOV capabilities

  capability name : sr_iov_devices possible values : count of such devices.

- NIC Capacity

  capability name : nic_capacity possible values : value with unit.

- TPM Support

  capability name : trusted_boot possible values : true, false

- Multi LUN support

  capability name : multi_lun possible values : true, false

- CDN (Consistent Device Name) Support

  capability name : cdn possible values : true, false

- VXLAN Capability

  capability name : vxlan possible values : true, false

- NV GRE Capability

  capability name : nv_gre_devices possible values : count of such devices

- NET FLOW Capability

  capability name : supports_net_flow possible values : true, false

- FlexFlash Capability

  capability name : flex_flash possible values : true, false

- UCS service-profile template name

  capability name : ucs_sp_template possible values : service profile template name

The properties which are already set will be overridden at reinvocation of inspect_hardware() except for NICs. If a port already exists, it will not create a new port for that MAC address. It will take care of adding as well as deleting of the ports for NIC changes [2]. Not all the capabilities are applicable to all Cisco UCS B/C/M-series server models. If a property is not applicable to the hardware, the property will not be added/updated in node.properties as capabilities. Inspection fetches only those capabilities applicable to the specific server model.

Inspection returns failure in the following cases:

- Failed to get basic properties.
- Failed to get capabilities, due to service-profile configuration errors.
- Communication errors with UCS Manager.

UCS specific module changes:

- Implement the InspectInterface method inspect_hardware().

Alternatives

These properties can be discovered manually outside the ironic and node.properties updated accordingly with the discovered properties.

Data model impact

None.

State Machine Impact

None

REST API impact

None.

Client (CLI) impact

None

RPC API impact

None.

Driver API impact

None.

Nova driver impact

None.

Ramdisk impact

N/A

Security impact

None.

Other end user impact

None.

Scalability impact

None.

Performance Impact

None.

Other deployer impact

None.

Developer impact

None.

Implementation

Assignee(s)

Primary assignee:
saripurigopi

Work Items

- Implementation of the InspectInterface class and its methods inspect_hardware(), validate() and get_properties().

Dependencies

- This feature is targeted for Cisco UCS B/C/M-series servers managed by UCS Manager 2.2(4b) or above. All the capabilities listed might not be available with older versions of UCS Manager (like 2.2(3b)).
- Depends on UcsSdk library.

Testing

Unit tests will be added conforming to ironic testing requirements, mocking UcsSdk. It will get tested on real hardware by UCS team with the available hardware models to the team.

Upgrades and Backwards Compatibility

No impact.

Documentation Impact

Hardware Inspection section will be added and updated accordingly in doc/source/drivers/ucs.rst.

References

1. UcsSdk library: * https://github.com/CiscoUCS/UcsSdk * https://pypi.org/project/UcsSdk

2. Introspect spec: * https://github.com/openstack/ironic-specs/blob/master/specs/kilo/ironic-node-properties-discovery.rst

5.15.117 Out-of-band RAID configuration using Cisco UCS drivers

https://bugs.launchpad.net/ironic/+bug/1526362

This blueprint proposes to implement out-of-band RAID configuration interface for Cisco UCS drivers. This implementation supports configuration of Cisco UCS Manager (UCSM) managed B/C/M-series servers.

Note

This specification has been retired as the vendor specific UCS driers are no longer available in Ironic. This was a result of the Open Source UcsSdk library no longer being maintained. Users may wish to explore use of the `redfish` driver, but it is unknown to the community if the UCS Redfish support has been extended to RAID support.

Problem description

Currently pxe_ucs and agent_ucs drivers dont support RAID configuration on UCSM managed servers.

Proposed change

It proposes implementing the RAID interface as described by the parent spec[1] for UCS drivers.

List of changes required:

- ucs.raid.RAIDInterface for RAID configuration operations

  The following methods will be implemented:

  - validate
  - create_configuration
  - delete_configuration

  validate() method validates the required UCS parameters for OOB RAID configuration. Also calls validate() of the super class to validate json schema. create_configuration and delete_configuration operations are implemented as asynchronous RAID configuration deployment operations by UCS drivers. UcsSdk/UCS-API asynchronously deploys the RAID configuration on the target node. UCS driver(s) sends the RAID configuration simultaneously to the target node when the operation is invoked, but UCS Manager

[1] New driver interface for RAID configuration: https://review.opendev.org/173214

would not deploy the configuration simultaneously on the target node. UCS Manager accepts the RAID configuration and deploys it as part of UCS Manager FSM deploy state. Hence there will be delay between, when the operation is invoked and when the RAID configuration is deployed. To know the deploy status, we need to query the FSM status using UcsSdk API. These methods return states.CLEANWAIT. New driver periodic task will be added to fetch the UCSM FSM status of these operations. This periodic task is enabled only if pxe_ucs, agent_ucs drivers are enabled in the conductor.

- RAID management changes:

  Controlling the RAID configuration is creating storage-profile ManagedObject and associating with the Server object in UCS Manager 2.4. Earlier version of UCSM requires to configure LocalDiskConfigPolicy and associate with respective service-profile ManagedObject. The service-profile information is captured as part of node driver_info properties. UcsSdk provides RAIDHelper interface, which actually creates the required policies specified above. UCS driver uses this interface and makes appropriate calls to create_config and delete_config operations.

Alternatives

Operator can change the RAID configuration manually whenever required after putting the node to MANAGEABLE state. But this has to be done for each node.

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

N/A

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

None

Developer impact

None

Implementation

Assignee(s)

Primary assignee: saripurigopi

Work Items

- Add UcsRAIDManagement inheriting from base.RAIDInterface for ucs drivers.
- Writing and unit-test cases for RAID interface of ucs drivers.
- Writing configuration documents.

Dependencies

- UcsSdk to support RAID configuration utility

Testing

Unit-tests will be implemented for RAID interface of ucs driver.

Upgrades and Backwards Compatibility

Adding RAID interface support for ucs drivers will not break any compatibility with either REST API or RPC APIs.

Documentation Impact

- Writing configuration documents.

References

5.15.118 Future of classic drivers

https://bugs.launchpad.net/ironic/+bug/1690185

This specification discusses the future of the classic drivers after the *Driver composition reform* was implemented in the Ocata cycle. Terminology here follows one of that specification.

Problem description

We do not want to maintain two approaches to building drivers long-term. It increases complexity of the source code (see e.g. driver_factory.py) and the amount of testing for 3rdparty CI.

Proposed change

The change covers several cycles:

Pike

In the Pike cycle hardware types and classic drivers will co-exist as equally supported ways of writing drivers.

- Dynamic drivers and all related API are considered stable and ready for production use.

  > **Note**
  >
  > In the Ocata release notes we called the related API additions experimental.

- No more classic drivers are accepted in tree.
- No new interfaces are added to the existing classic drivers.
- No interface implementations are changed in the existing classic drivers.
- We *recommend* the vendors to provide hardware types analogous to their existing classic drivers. 3rd party CI should provide the complete coverage of all supported boot, deploy, management and power interface combinations. Its up to the vendors to decide whether to use classic drivers or hardware types to achieve that.

Queens

In the Queens cycle we will deprecate classic drivers.

- We will *require* the vendors to provide hardware types analogous to their existing classic drivers. It is up to the vendors to choose the combination of interfaces to support. It will be *recommended*, however, to keep support for standard deploy and inspect interface implementations, if possible.

- 3rd party CI will have to cover all hardware types and all supported combinations of the boot, deploy, management and power interfaces. 3rd party CI will be able to stop covering supported classic drivers, when their functionality is covered through hardware types.
- The classic drivers mechanism will be deprecated, and loading any classic driver (in-tree or out-of-tree) will result in a deprecation warning. The `enable_drivers` configuration option will be also deprecated.

> **Note**
>
> In the Queens release we will continue running regular CI against classic drivers still.

- Existing (in-tree) classic drivers will only receive critical bug fixes as related to the classic interface (i.e. they will still be affected by fixes in the interface implementations they share with hardware types).
- Most of the upstream CI will run on the dynamic drivers (`ipmi`, `snmp` and `redfish`). The standalone job will provide coverage for classic drivers. Grenade will be testing switch from classic drivers (e.g. `pxe_ipmitool`) to hardware types (e.g. `ipmi`).
- Deprecate `-t/--type` argument to driver listing commands in `python-ironicclient`.

> **Note**
>
> Deprecating or removing the `type` argument to the driver listing API is outside of the scope of this proposal.

- Extend the upgrade documentation to contain a full mapping between supported classic drivers and associated combination of a hardware type and hardware interfaces. Explicitly mention classic drivers that will not receive a new counterpart per vendor decision, and which replacement is recommended for such drivers.
- Update the whole documentation to only mention hardware types, except for the driver-specific documentation and the upgrade documentation bit explained above.
- Provide automatic migration to hardware types as part of the `online_data_migration` command - see *Automatic migration*.

> **Note**
>
> We decided to not provide any automatic migration on the API level in the node create and update API. Doing so would require us to maintain mapping between classic drivers and corresponding hardware types/interfaces foreever. It also may be confusing for operators, if, for example, the result of the node creation request differs from the outcome.

Rocky

In the Rocky release the support for classic drivers is removed.

- Remove all in-tree classic drivers.
- Remove support for loading classic drivers from driver_factory.py.

- Remove the `enable_drivers` configuration option.
- Remove CI coverage for classic drivers.
- Remove `-t/--type` argument to driver listing commands in `python-ironicclient`.
- Update the driver listing API to always return an empty result when `classic` type is requested.

Automatic migration

To simplify transition for operators, make `online_data_migration` in the Queens release automatically update nodes.

- Extend BaseDriver with a new class method:

```
@classmethod
def to_hardware_type(cls):
    """Return corresponding hardware type and hardware interfaces.

    :returns: a tuple with two items:

        * new driver field - the target hardware type
        * dictionary containing interfaces to update, e.g.
          {'deploy': 'iscsi', 'power': 'ipmitool'}
    """
```

  For example, for the `agent_ipmitool` driver:

```
@classmethod
def to_hardware_type(cls):
    if CONF.inspector.enabled:
        inspect_interface = 'inspector'
    else:
        inspect_interface = 'no-inspect'

    return 'ipmi', {'boot': 'pxe',
                    'deploy': 'direct',
                    'inspect': inspect_interface,
                    'management': 'ipmitool',
                    'power': 'ipmitool',
                    'raid': 'agent'}
```

- Update the `online_data_migrations` to accept options for migrations in the form of `--option <MIGRATION NAME><KEY>=<VALUE>`. They will be passed as keyword arguments to the migration matching the provided name.
- Update the `online_data_migrations` command with a new migration `migrate_to_harware_types`. It will accept one option `reset_unsupported_interfaces`, which is a boolean value with the default of `False`. The migration will do the following:

  1. Load classes for all classic drivers in the `ironic.drivers` entrypoint (but do not instantiate them).
  2. For each classic driver:
     1. Calculate required changes using `DriverClass.to_hardware_type`.

Missing interfaces, other than `boot`, `deploy`, `management` and `power`, are defaulted to their no-op implementations (`no-***`).

> **Note**
>
> We consider `boot`, `deploy`, `management` and `power` mandatory, as they do not have a no-op implementation.

2. If the hardware type is not in `enabled_hardware_types`, issue a and skip all nodes with this classic driver.
3. If any interface is not enabled (not in `enabled_***_interfaces`):
    1. if this interface is one of `boot`, `deploy`, `management` or `power`, or if `reset_unsupported_interfaces` is `False`, issue a warning and skip the nodes.
    2. otherwise try again with resetting the interface to its no-op implementation (`no-***`).
4. Update the node record in the database.

> **Note**
>
> Due to idempotency of the migrations, operators will be able to re-run this command after fixing the warnings to update the skipped nodes.

- In the **Rocky** cycle, update the `dbsync` command with a check that no nodes are using classic drivers. As the list of classic drivers will not be available at that time (they will be removed from the tree), maintain the list of classic driver names that used to be in tree and check nodes against this list. Remove this check in the release after Rocky.

Alternatives

- Keep classic drivers forever. Complicates maintenance for unclear reasons.
- Start deprecation in the Pike cycle. We wanted to have at least one cycle where hardware types are fully supported before we jump into deprecation. Also, in this case we will have to rush the vendors into creating and supporting their hardware types before end of Pike.

Data model impact

None

State Machine Impact

None

REST API impact

Due to the way we designed *Driver composition reform*, dynamic drivers look very similar to the classic drivers for API point of view.

We could deprecate the `type` argument to the driver listing API. However,

1. API deprecations are hard to communicate,
2. due to API versioning, we will still have to support it forever.

Thus, this specification does not propose deprecating anything in the API.

Client (CLI) impact

ironic CLI

Deprecate `-t` argument to the `driver-list` command in the Queens cycle and remove it in Rocky.

openstack baremetal CLI

Deprecate `--type` argument to the `baremetal driver list` command in the Queens cycle and remove it in Rocky.

RPC API impact

None

Driver API impact

- In the Queens release, all classic drivers will behave as if they had `supported = False`.
- In the Rocky release, support for loading classic drivers will be removed. `BaseDriver` will be merged with `BareDriver`, code in driver_factory.py will be substantially simplified.

Nova driver impact

None

Ramdisk impact

None

Security impact

None

Other end user impact

Users of Ironic will have to switch their deployment to hardware types before upgrading to Rocky.

Scalability impact

None

Performance Impact

None

Other deployer impact

See *Upgrades and Backwards Compatibility*.

Developer impact

Out-of-tree classic drivers will not work with the Rocky release of Ironic.

Implementation

Assignee(s)

Primary assignee:
Dmitry Tantsur (IRC: dtantsur, LP: divius)

Work Items

See *Proposed Change* for the quite detailed breakdown.

Dependencies

None

Testing

Starting with the Queens release, our CI will mainly test hardware types.

We will modify the Grenade job testing Pike -> Queens upgrade to switch from `*_ipmitool` to `ipmi` during the upgrade.

Upgrades and Backwards Compatibility

Removing the drivers and the classic driver mechanism is going to be a breaking change and has to be communicated accordingly.

Operators will have to enable appropriate hardware types and hardware interfaces in the Queens release.

Documentation Impact

The upgrade guide will be updated to explain moving from classic drivers to hardware types with a examples and a mapping between old and new drivers.

References

5.15.119 Change CLI default API version to latest known ironic API version

https://bugs.launchpad.net/python-ironicclient/+bug/1671145

When using ironic CLI or the OpenStack CLI (OSC), ironic API version 1.9 is used by default. This spec proposes raising the default ironic API version used in both CLIs to the latest compatible API version.

Problem description

Currently, ironic CLI and the baremetal OSC plugin default to using an API version that dates to the Liberty release.[1]

[1] https://docs.openstack.org/developer/ironic/dev/webapi-version-history.html

This means that any new user using the CLI who doesnt specify the version can only use features that are almost 2 years old as of the time of this writing. This limits discoverability and usability of new features.

Also, if we ever bump up the minimum supported API version in the API, we will have to deal with raising the minimum CLI version anyway.

Proposed change

- For the Pike cycle, a warning message will be printed when a user runs either CLI without specifying the version, indicating that the CLI will soon default to using the latest compatible API version. The proposed wording for the OSC plugin is as follows:

  > You are using the default API version of the OpenStack CLI baremetal (ironic) plugin. This is currently API version 1.9. In the future, the default will be the latest API version understood by both API and CLI. You can preserve the current behavior by passing the os-baremetal-api-version argument with the desired version or using the OS_BAREMETAL_API_VERSION environment variable.

  A similar wording will be used for the `ironic` tool:

  > You are using the default API version of the `ironic` CLI tool. This is currently API version 1.9. In the future, the default will be the latest API version understood by both API and CLI. You can preserve the current behavior by passing the ironic-api-version argument with the desired version or using the IRONIC_API_VERSION environment variable.

  If the user wishes to continue using API version 1.9, they should specify so on the command line.

- During the Queens cycle, the default API version used by the CLI will change to the latest version of the API compatible with the CLI. If only the major version is specified (for example, `--os-baremetal-api-version=1`), the latest compatible version of that major version will be used (e.g. 1.32 if thats the latest 1.XX version).

  The latest compatible version will be determined via version negotiation between the CLI and ironic. The CLI will first make a request to the root endpoint of the ironic API, which returns the version of the ironic API.[2] If the ironic API version is lower than the maximum version the client is compatible with, the CLI will use the version running on the API service to ensure compatibility. Otherwise, the maximum ironic API version that can be handled by the client will be used.

  Note

  We may deprecate the `ironic` tool in the Queens cycle. If this happens, we may decide to skip applying this change to this tool, and proceed with the deprecation instead. This is subject to a separate spec.

Changes to the client library are out of scope for this spec.

This change was discussed in detail at the Pike PTG.[3]

[2] https://etherpad.openstack.org/p/ironic-pike-ptg-operations
[3] https://developer.openstack.org/api-ref/baremetal/#list-api-versions

Alternatives

- We could periodically bump the default API version used by the CLI. This has the disadvantage of being unpredictable from a user standpoint and doesnt solve the discoverability issue for the latest features. Its also not ideal for maintainability of the CLI codebase.
- We could keep the default as 1.9 and always log a warning, without ever changing the default to the latest version. This doesnt solve the ease of use or discoverability issues, and it permanently adds an extra annoyance to users who dont wish to specify a version every time the CLI is invoked.
- We could just pass `latest` to the API when no version is provided. However, this approach would lead to potentially broken behavior if we ever land a breaking change to our API.

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

ironic CLI

A warning will be logged when ironic CLI is invoked without specifying `--ironic-api-version`, indicating that the default will soon be the latest compatible API version.

After the deprecation period, the default will be changed accordingly, assuming that ironic CLI itself is not deprecated beforehand.

openstack baremetal CLI

A warning will be logged when the OSC baremetal plugin is invoked without specifying `--os-baremetal-api-version`, indicating that the default will soon be the latest compatible API version.

After the deprecation period, the default will be changed accordingly.

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

None

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

None

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
dtantsur

Other contributors:
mariojv

Work Items

- Add the warning message
- After the standard deprecation period (release boundary or 3 months, whichever is longer), change the default API version used by the CLI.

Dependencies

None

Testing

Appropriate unit and functional testing will be added.

Upgrades and Backwards Compatibility

If a user has scripts or tooling that use the CLI without specifying the version, those will need to be updated to specify the version.

Documentation Impact

None. Appropriate release notes will be added.

References

5.15.120 Collect logs from agent ramdisk

https://bugs.launchpad.net/ironic/+bug/1587143

This spec adds support for retrieving the deployment system logs from agent ramdisk.

Problem description

Currently we do not have any documented mechanism to automatically retrieve the system logs from Bootstrap. Having access to agent ramdisk logs may be highly required especially after deployment failure. Collecting logs on remote server is common deployment thing. There are a lot of proprietary, opensource, standardized technologies that solve this task. Lets walk through linux natively supported:

1. Syslog: is the standardized protocol for message logging, described in RFC. All distros supports syslog in different implementations `rsyslogd`, `syslogd`, `syslog-ng`
2. Systemd journal-remote: systemd tool to send/receive journal messages over the network.

Proposed change

The proposed implementations is about adding documentation that helps Operators to build agent ramdisk with collecting logs feature. On top of syslog and systemd journal-remote technologies.

Changes in IPA

A new kernel parameter is added `use_syslog`.

1. `use_syslog` (Boolean): Whether IPA should send the logs to syslog or not. Defaults to False.

Changes in Ironic

New pxe_append parameters are added `use_syslog` and `syslog_server` in `[agent]` section.

1. `use_syslog` boolean): Whether IPA should send the logs to syslog or not. Defaults to False.
2. `syslog_server` (string): IP address or DNS name of syslog server to send system logs to. Defaults to IP address of ironic-conductor node. The variable may be used to configure `rsyslogd` or `syslog` or any other syslog server implementation.

Changes in agent ramdisk

Bootstrap should support remote logging via syslog/systemd. All agent ramdisk changes are related to logging tool configuration (rsyslogd, syslogd, journal-remote). The logging service should be started right after networking service to make sure that we send all logs from OpenStack services. On `Collector` side (ironic-conductor) we receive messages from `Originator` (agent ramdisk) and place the according to local rules. It may be directory on the filesystem: /var/log/remote/ironic/<node_uuid>/<log_filename>

Alternatives

Implement logs collecting mechanism in Ironic and IPA as it described in https://review.opendev.org/323511

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

Support will be added to ramdisk build tools, such as `diskimage-builder` and image building scripts under IPA repository to build ramdisks with remote logging support.

Security impact

None.

Other end user impact

None.

Scalability impact

None

Performance Impact

None

Other deployer impact

Deployer chooses which mechanism for collecting logs should be used. General changes to agent ramdisk described at *Changes in agent ramdisk*

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
lucasagomes <lucasagomes@gmail.com> vsaienko <vsaienko@mirantis.com>

Other contributors:

Work Items

- Add documentation that describes how to configure `Collector` on ironic-conductor, and `Originator` in ironic agent ramdisk on top of rsyslogd or syslogd or journal-remote.
- Add an examples of building most popular images (coreos, tinyipa) with syslog support. New `dib` element will be added as well.
- Integrate configuring `Collector` on devstack host.

Dependencies

None

Testing

Collecting logs from agent ramdisk will be configured on gate jobs. The directory with logs from agent ramdisk will be archived and available in the jobs artefacts.

Upgrades and Backwards Compatibility

None.

Documentation Impact

Documentation will be provided about how to configure syslog `Collector` to receive and store logs from `Originator`.

References

None.

5.15.121 Collect system logs from IPA

https://bugs.launchpad.net/ironic/+bug/1587143

This spec adds support for retrieving the deployment system logs from IPA.

Problem description

We currently have no mechanism to automatically retrieve the system logs from the IPA deploy ramdisk. Having access to the logs may be very useful, especially when troubleshooting a deployment failure. Currently, there are a few ways to get access to the logs in the ramdisk, but they are manual, and sometimes it is not desirable to enable them in production. The following points describe two of them:

1. Have a console session enabled for the node being deployed.

   While this works, its tricky because the operator needs to figure out which node was picked by the scheduler and enable the console for it. Also, not all drivers have console support.

2. Disable powering off a node upon a deployment failure.

   Operators could disable powering off a node upon a deployment failure but this has some implications:

   a. It does not work in conjunction with Nova. When the instance fails to be provisioned nova will invoke destroy() and the Ironic virt driver will then force a power off on that node.

   b. Leaving the nodes powered on after the failure is not desirable in some deployments.

Proposed change

The proposed implementation consists in having Ironic retrieve the system logs from the deploy ramdisk (IPA) via its API and then upload it to Swift or save it on the local file-system of that conductor (for standalone-mode users).

Changes in IPA

A new `log` extension will be added to IPA. This extension will introduce a new synchronous command called `collect_system_logs`. By invoking this command IPA will then tar, gzip and base64 encode the system logs and return the resulting string to the caller.

Since we do support different base OSs for IPA (e.g Tiny Core Linux, Fedora, Debian) we need different ways to find the logs depending on the system. This spec proposes two ways that should be enough for most of the distros today:

1. For distributions using `systemd`, all system logs are available via `journald`. IPA will then invoke the `journalctl` command and get the logs from there.

2. For other distributions, this spec proposes retaining all the logs from */var/log* and the output of the `dmesg` command.

The logs from all distributions independent of the init system, will also contain the output of the following commands files: `ps`, `df`, and `iptables`.

Changes in Ironic

New configuration options will be added to Ironic under the `[agent]` group:

1. `deploy_logs_collect` (string): Whether Ironic should collect the deployment logs or not. Valid options are: always, on_failure or never. Defaults to on_failure.
2. `deploy_logs_storage_backend` (string): The name of the storage backend where the response file will be stored. One of the two: local or swift. Defaults to local.
3. `deploy_logs_local_path` (string): The path to the directory where the logs should be stored, used when the `deploy_logs_storage_backend` is configured to **local**. Defaults to `/var/log/ironic/deploy`.
4. `deploy_logs_swift_container` (string): The name of the Swift container to store the logs, used when the `deploy_logs_storage_backend` is configured to **swift**. Defaults to *ironic_deploy_logs_container*.
5. `deploy_logs_swift_days_to_expire` (integer): Number of days before a log object is marked as expired in Swift. If None, the logs will be kept forever or until manually deleted. Used when the `deploy_logs_storage_backend` is configured to **swift**. Defaults to *30 days*.

Note

When storing the logs in the local file-system Ironic wont be responsible for deleting the logs after a certain time. Its up to the operator to configure an external job to do it, if wanted.

Depending on the value of the `deploy_logs_collect` Ironic will invoke `log.collect_system_logs` as part of the deployment of the node (right before powering it off or rebooting). For example, if `deploy_logs_collect` is set to **always** Ironic will collect the logs independently of the deployment being a success or a failure; if it is set to **on_failure** Ironic will collect the logs upon a deployment failure; if it is set to **never**, Ironic never collect the deployment logs.

When the logs are collected, Ironic should decode the base64 encoded tar.gz file and store it according to the `deploy_logs_storage_backend` configuration. All log objects will be named with the following pattern: *<node-uuid>[_<instance-uuid>]_<timestamp yyyy-mm-dd-hh:mm:ss>.tar.gz*. Note that, `instance_uuid` is not a required field for deploying a node when Ironic is configured to be used in **standalone** mode so, if present it will be appended to the name.

When using Swift, operators can associate the objects in the container with the nodes in Ironic and search for the logs of a specific node using the `prefix` parameter, for example:

```
$ swift list ironic_deploy_logs_container -p 5e9258c4-cfda-40b6-86e2-
↪e192f523d668
5e9258c4-cfda-40b6-86e2-e192f523d668_0c1e1a65-6af0-4cb7-a16e-8f9a45144b47_
↪2016-05-31_22:05:59
5e9258c4-cfda-40b6-86e2-e192f523d668_db87f2c5-7a9a-48c2-9a76-604287257c1b_
↪2016-05-31_22:07:25
```

Note

This implementation requires the network to be setup correctly, otherwise Ironic will not be able to contact the IPA API. When debugging such problems, the only action possible is to look at the consoles of the nodes to see some logs. This method has some caveats: see the *Problem description* for more information.

Note

Neither Ironic or IPA will be responsible for sanitizing any logs before storing them. First because this spec is limited to collecting logs from the deployment only and at this point the tenant wont have used the node yet. Second, the services generating the logs should be responsible for masking secrets in their logs (like we do in Ironic), if not, it should be considered a bug.

Alternatives

Since we already provide ways of doing that via accessing the console or disabling the powering off the nodes on failures, there are few alternatives left for this work.

The current proposed solution could be extended to fit more use cases beyond what this spec proposes. For example, instead of uploading it to Swift or storing it in the local file-system, Ironic could upload it to a HTTP/FTP server.

As briefly described at *Changes in IPA* the method to collect the logs could be extended to include more logs and output of different commands that are useful for troubleshooting.

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

None

Security impact

None.

As a note, credentials **are not** passed from Ironic to the deploy ramdisk. The `ironic-conductor` service, which already holds the Swift credentials, is the one responsible for uploading the logs to Swift.

Other end user impact

None

Scalability impact

None

Performance Impact

The node will stay a little longer in the `deploying` provision state while IPA is collecting the logs, if enabled.

Other deployer impact

None

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
lucasagomes <lucasagomes@gmail.com>

Other contributors:

Work Items

- Add the new `log` extension and `collect_system_logs` method in IPA.
- Add the new configuration options described in the *Changes in Ironic* section.
- Invoke the new `log.collect_system_logs` method in IPA as part of the deployment and store the response file according to the `deploy_logs_storage_backend` configuration option (if enabled).

Dependencies

None

Testing

Unittests will be added.

Upgrades and Backwards Compatibility

None.

As a note, when using an old IPA ramdisk which does not support the new `log.collect_system_logs` command Ironic should handle such exception and log a warning message to the operator if `deploy_logs_collect` is set to **always** or **on_failure**.

Documentation Impact

Documentation will be provided about how to configure Ironic to collect the system logs from the deploy ramdisk.

References

None.

5.15.122 Conductor/node grouping affinity

https://storyboard.openstack.org/#!/story/2001795

This spec proposes adding a `conductor_group` property to nodes, which can be matched to one or more conductors configured with a matching `conductor_group` configuration option, to restrict control of those nodes to these conductors.

Problem description

Today, there is no way to control the conductor-to-node mapping. This is desirable for a number of reasons:

- An operator may have an Ironic deployment that spans multiple sites. Without control of this mapping, images may be pulled over WAN links. This causes slower deployments and may be less secure.
- Similarly, an operator may want to map nodes to conductors that are physically closer to the nodes in the same site, to reduce the number of network hops between the node and the conductor. A prime example of this would be to place a conductor in each rack, reducing the path to only go through the top-of-rack switch.
- A deployer may have multiple networks for out-of-band control, that must be completely isolated. This feature would allow isolating a conductor to a single out-of-band network.
- A deployer may have multiple physical networks that not all conductors are connected to. By configuring the mapping correctly, conductors can manage only the nodes which they can communicate with. This is described further in another RFE.[0]

Proposed change

We propose adding a `conductor_group` configuration option for the conductor, which is a single arbitrary string specifying some grouping characteristic of the conductor.

We also propose a `conductor_group` field in the node record, which will be used to map a node to a conductor. This matching will be done case-insensitive, to make things a bit easier for operators.

A blank `conductor_group` field or config is the default. A conductor without a group can only manage nodes without a group, and a node without a group can only be managed by a conductor without a group.

The hash ring will need to be modified to take grouping into account, as described below in the *RPC API Impact* section.

Alternatives

Another RFE[1] proposes a complex system of hard and soft affinity, affinity and anti-affinity, and scoring of placement to a conductor with multiple tags. This is quite complex, and I dont believe well get it done in the short term. Completing this more basic work doesnt block this more complex work, and so we should take it one step at a time.

Data model impact

A `conductor_group` field will be added to the nodes table, as a `VARCHAR(255)`. This will have a default of `""`, or the empty string. This string will be used in the hash ring calculation, so theres no sense in defaulting to `NULL`.

A `conductor_group` field will also be added to the conductors table, also as a `VARCHAR(255)`. This will also have a default of `""`, or the empty string. This will be used to build the hash ring to look up where nodes should land.

State Machine Impact

None.

REST API impact

The `conductor_group` field of the node will be added to the node object in the REST API, with a microversion as usual. It will be allowed in POST and PATCH requests. As with the database, it will be restricted to 255 characters. There must be a conductor in that group available, as the conductor services node creation and updates, and is selected via the hash ring.

Its worth noting that we would like to expose the grouping of conductors via the REST API eventually. However, the best way to do this isnt immediately clear, so we leave it outside the scope of this spec for now. Another RFE[3] proposes a service management API that may be a good fit.

Client (CLI) impact

ironic CLI

None, its deprecated.

openstack baremetal CLI

The `conductor_group` field for a node will be exposed in the client output, and added to the `node create` and `node set` commands.

RPC API impact

This will affect which conductor is the destination for RPC calls corresponding to a given node, however wont have a direct effect on the RPC API itself.

The hash ring will change such that the internal keys for the hash ring will now be of the structure `"$conductor_group:$drivername"`. A colon (`:`) is used as the separator between the two, to eliminate conflicts between conductor groups and drivers or hardware types. For example, an `agent_ilo` key with no separator could mean a node with no group and the `agent_ilo` driver, or it could mean a node with group `agent_` using the `ilo` hardware type. To handle upgrades, hash ring keys will be built without the conductor group while the service is pinned to a version before this feature, and built with the conductor group when the service is unpinned or pinned to a version after this feature is implemented.

We handle upgrades by ignoring grouping for services which have a pin in the RPC version that is less than the release with this feature. Once everything is upgraded and unpinned, we begin using the grouping tags configured.

Operators should leave a sufficient number of conductors available without a grouping tag configured to run the cluster, until nodes can be configured with the grouping tag. Any nodes without a grouping tag will only be managed by conductors without a grouping tag.

Driver API impact

Hash ring generation and lookup will include the grouping tag, as specified above in the *RPC API Impact* section.

Nova driver impact

This change is transparent to Nova.

Ramdisk impact

None.

Security impact

No direct impact; however this provides another mechanism for securing a deployment by enabling logical infrastructure segregation.

Other end user impact

None.

Scalability impact

None.

Performance Impact

None.

Other deployer impact

Deployers that wish to use this feature will need to manage the process of labeling conductors and nodes to enable it, which may be a non-trivial task.

Developer impact

None.

Implementation

Assignee(s)

Primary assignee:
jroll

Other contributors:
dtantsur

Work Items

- Add database fields.
- Add conductor config and populate conductor DB field.
- Change the hash ring calculation, and bump the RPC API so that we can pin during upgrades.
- Add fields to the node and conductor objects.
- Make the REST API changes.
- Update the client library/CLI.
- Document the feature.

Dependencies

None.

Testing

Unit tests should be sufficient, as thats how we test our hash ring now. Its difficult to test this with Tempest without exposing conductor grouping via the REST API.

Upgrades and Backwards Compatibility

This is described in the *RPC API Impact* section.

Documentation Impact

This should be documented in the install guide and admin guide.

References

[0] https://storyboard.openstack.org/#!/story/1734876

[1] https://storyboard.openstack.org/#!/story/1739426

[2] Notes from the Rocky PTG session:
https://etherpad.openstack.org/p/ironic-rocky-ptg-location-awareness

[3] https://storyboard.openstack.org/#!/story/1526759

5.15.123 Add notifications about resources CRUD and node states

https://bugs.launchpad.net/ironic/+bug/1606520

This spec proposes addition of new notifications to ironic: CRUD (create, update, or delete) of resources and node state changes for provision state, maintenance and console state.

Problem description

Resource indexation services like Searchlight[1] require notifications about creation, update or deletion of a resource. Currently CRUD notifications are not implemented in ironic. Creating an efficient plugin for Searchlight is impossible without these notifications. Ironic node notifications for provision state, maintenance and console state also could be used by Searchlight plugin in order to keep Searchlights index of ironic resources up-to-date.

Apart from searchlight, there is a use case of monitoring service, that caches all notification payloads along with event type, like start/end/error/<etc> and an operator can query this service to see if ironic is behaving properly. For example, if there are much more start notifications for node create, than there are end notifications, it may mean that the database is not behaving properly, or messaging is having a hard time delivering messages between API and conductor. That is a separate case from searchlight: searchlight for example does not need to know the payload of the node create start notification, as there is no actual node yet, but for monitoring purposes, it may be useful.

Proposed change

As a general note for all CRUD notifications, `*.start` and `*.error` event payloads will be ignored by Searchlight, as in both cases it would mean that resource representation has not changed, or in case of `*create*` notifications, that the resource was not created.

Node CRUD notifications

The following event types will be added:

- baremetal.node.create.start;
- baremetal.node.create.end;
- baremetal.node.create.error;
- baremetal.node.update.start;
- baremetal.node.update.end;

[1] https://wiki.openstack.org/wiki/Searchlight

- baremetal.node.update.error;
- baremetal.node.delete.start;
- baremetal.node.delete.end;
- baremetal.node.delete.error.

Priority level - INFO or ERROR (for error status). Payload contains all fields from base `NodePayload` with additional fields: `chassis_uuid`, `instance_info`, `driver_info`. Secrets in the node fields will be masked. `raid_config` and `target_raid_config` fields are excluded because they can contain low-level disk and vendor information. If/when there is a use case for them, they can be added in the future. All these notifications will be implemented at the API level.

Port CRUD notifications

The following event types will be added:

- baremetal.port.create.start;
- baremetal.port.create.end;
- baremetal.port.create.error;
- baremetal.port.update.start;
- baremetal.port.update.end;
- baremetal.port.update.error;
- baremetal.port.delete.start;
- baremetal.port.delete.end;
- baremetal.port.delete.error.

Priority level - INFO or ERROR (for error status). Payload contains these fields: `uuid`, `node_uuid`, `address`, `extra`, `local_link_connection`, `pxe_enabled`, `created_at`, `updated_at`. These notifications will be implemented at the API level. In addition, baremetal.port.create.* will be emitted by the ironic-conductor service when driver creates a port (examples are[2] and[3]).

Chassis CRUD notifications

The following event types will be added:

- baremetal.chassis.create.start;
- baremetal.chassis.create.end;
- baremetal.chassis.create.error;
- baremetal.chassis.update.start;
- baremetal.chassis.update.end;
- baremetal.chassis.update.error;
- baremetal.chassis.delete.start.

[2] https://github.com/openstack/ironic/blob/2c76da5f437c5fc2f4022e8705e74fed0a46bebb/ironic/drivers/modules/irmc/inspect.py#L177

[3] https://github.com/openstack/ironic/blob/2c76da5f437c5fc2f4022e8705e74fed0a46bebb/ironic/drivers/modules/ilo/inspect.py#L56

- baremetal.chassis.delete.end.
- baremetal.chassis.delete.error;

Priority level - INFO or ERROR (for error status). Payload contains these fields: `uuid`, `extra`, `description`, `created_at`, `updated_at`. All these notifications will be implemented at the API level.

Node provision state notifications

Will be implemented via TaskManager methods (and emitted by the ironic-conductor service).

Types of events for node provision state:

- baremetal.node.provision_set.start;
- baremetal.node.provision_set.end;
- baremetal.node.provision_set.error;
- baremetal.node.provision_set.success.

Types of state changing in ironic and corresponding events:

- Start transition, spawning a working thread: start notification with INFO level.
- End transition, cleaning `target_provision_state`: end notification with INFO level.
- Error events processing: error notification with ERROR level.
- Change `provision_state` without starting a worker that is not end or error: success notification with INFO level. Examples are DEPLOYING <-> DEPLOYWAIT, AVAILABLE -> MANAGEABLE.

Payload contains all fields from base `NodePayload` with additional fields: `instance_info`, `previous_provision_state`, `previous_target_provision_state`, `event` (FSM event that triggered the state change). To efficiently use the provision state notifications all related node changes (like setting of `last_error`, `maintenance`) should be done before event processing.

Node maintenance notifications

The following event types will be added:

- baremetal.node.maintenance_set.start;
- baremetal.node.maintenance_set.end;
- baremetal.node.maintenance_set.error.

Priority level - INFO or ERROR (for error status). Payload contains all fields from base `NodePayload`. All these notifications will be implemented at the API level and reflect maintenance changes to a node due to a user request. There wont be any explicit node maintenance notifications for maintenance changes done internally by ironic. Since these internal changes occur as a result of trying to change the nodes state (e.g. provision, power), one of the other notifications that is emitted will cover these internal maintenance changes.

Node console notifications

The following event types will be added:

- baremetal.node.console_set.start;
- baremetal.node.console_set.end;
- baremetal.node.console_set.error;
- baremetal.node.console_restore.start;
- baremetal.node.console_restore.end;
- baremetal.node.console_restore.error.

`console_set` action is used when start or stop console is initiated via API request, `console_restore` action is used when `console_enabled` flag is already enabled in the DB for node and console restart via driver is required (due to dead or restarted ironic-conductor process). Priority level - INFO or ERROR (for error status). Payload contains all fields from base `NodePayload`. All these notifications will be implemented in the ironic-conductor, because setting of a nodes console is an asynchronous request, so ironic-conductor can easily emit notifications for the start/end of the change.

Alternatives

Periodically polling ironic resources via API.

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

None

Security impact

None

Other end user impact

None

Scalability impact

If notifications are enabled, they can create high load on the message bus during node deployments on large environments.

Performance Impact

None

Other deployer impact

Deployers should set already existing `notification_level` config options properly.

Developer impact

- If developer creates resources in the driver, proper notification should be emitted.
- For provision state change all related node updates should be done before event processing.

Implementation

Assignee(s)

Primary assignee:

- yuriyz

Other contributors:

- vdrok
- mariojv

Work Items

- Implement node provision state change notifications.
- Implement CRUD notifications and node maintenance notifications.
- Implement console notifications.

- Add notifications to the current ironic code that creates resources in the drivers.
- Fix ironic code with node updates after event processing.

Dependencies

Patch with base `NodePayload`[4].

Testing

Unit tests will be added.

Upgrades and Backwards Compatibility

None

Documentation Impact

New notifications feature will be documented.

References

5.15.124 Deploy Templates

https://storyboard.openstack.org/#!/story/1722275

This builds on the deployment steps spec, such that it is possible to request a particular set of deployment steps by requesting a set of traits.

When Ironic is used with Nova, this allows the end user to pick a Flavor, that requires a specific set of traits, that Ironic turns into a request for a specific set of deployment steps in Ironic.

Problem description

One node can be configured in many ways. Different user workloads require the node to be configured in different ways. As such, operators would like a single pool of hardware to be reconfigured to match each users requirements, rather than attempting to guess how many of each configurations are required in advance.

In this spec, we are focusing on reconfiguring regular hardware to match the needs of each users workload. We are not considering composable hardware.

When creating a Nova instance with Ironic as the backend, there is currently no way to influence the deployment steps used by Ironic to provision the node.

Example Use Case

Consider a bare metal node that has three disks. Different workloads may require different RAID configurations. Operators want to test their hardware and determine the specific set of RAID configurations that work best, and offer that choice to users of Nova, so they can pick the configuration that best matches their workload.

[4] https://review.opendev.org/#/c/321865/

Proposed change

Context: Deployment Steps Framework

This spec depends on the deployment steps framework spec. The deployment steps framework provides the concept of a deployment step that may be executed when a node is deployed.

It is assumed that there is a default set of steps that an operator configures to happen by default on every deploy. Each step task has a priority defined, to determine the order that that step runs relative to other enabled deploy steps. Configuration options and hard-coded priorities define if a task runs by default or not. The priority says when a task runs if the task should run during deployment.

Deploy Templates

This spec introduces the concept of a `Deploy Template`. It is a mapping from a valid trait name for the node to one or more Deployment Steps and all the arguments that should be given to those deployment steps. There is a new API added to Ironic for Operators to specify these `Deploy Templates`.

To allow a `Deploy Template` for a given Ironic node, you add the trait name of the deploy template to the `traits` list on each Ironic node that needs the deploy template enabled. The validation of the node must fail if any of the enabled `Deploy Templates` are not supported on that node.

It is worth noting that all traits set on the node are synchronised to Novas placement API. In turn, should a user request a flavor that requires a trait (that may or may not map to a deploy template) only nodes that have the trait set will be offered as candidates by Novas Placement API.

To request the specified `Deploy Template` when you provision a particular Ironic node, the corresponding trait is added to a list in `node.instance_info` under the key of `traits`. This is what Novas ironic virt driver already does for any required trait in the flavors extra specs for that instance. Again, Ironic already validates that the traits in `node.instance_info` are a subset of the traits that the Operator has set on the Ironic node, via the new node-traits API.

During the provisioning process Ironic checks the list of traits set in `node.instance_info` and checks if they match any `Deploy Templates`. The list of matching `Deploy Templates` are then used to extend the list of deployment steps for this particular provision operation. As already defined in the deployment steps framework, this is then combined with the list of deployment steps from what is configured to happen by default for all builds.

The order in which the steps are executed will be defined by the priority of each step. If the code for a deploy step defines a deploy priority of 0, and that is not changed by a configuration option, that deploy step does not get executed by default. If a deploy template specifies a priority (this is required if the code has a default priority of 0), this overrides both the code default and any configuration override.

It is acceptable to request the same deploy step more than once. This could be done to execute a deploy step multiple times with different arguments, for example to partition multiple disks.

Any deploy step in a requested deploy template will override the default arguments and priority for that deploy step. 0 is the only priority override that can be set for any of the core deploy steps, i.e. you can only disable the core step, you cant change the order of its execution.

Trait names used in Deploy Templates should be unique - no two deploy templates should specify the same trait name.

In summary, you can use traits to specify additional deploy steps, by the mapping between traits and deploy steps specified in the new `Deploy Templates` API.

Current Limitations

When mapping this back to Nova integration, currently there would need to be a flavor for each of these combinations of traits and resource_class. Longer term Nova is expected to offer the option of a user specifying an override trait on a boot request, based on what the flavor says is possible. This spec has no impact on the Nova ironic virt driver beyond what is already implemented to support node-traits.

Example

Lets now look at how we could request a particular deploy template via Nova.

`FlavorVMXMirror` has these extra specs:

- `resource:CUSTOM_COMPUTE_A = 1`
- `trait:CUSTOM_CLASS_A = required`
- `trait:CUSTOM_BM_CONFIG_BIOS_VMX_ON = required`
- `trait:CUSTOM_BM_CONFIG_RAID_DISK_MIRROR = required`

`FlavorNoVMXStripe` has these extra specs:

- `resource:CUSTOM_COMPUTE_A = 1`
- `trait:CUSTOM_CLASS_A = required`
- `trait:CUSTOM_BM_CONFIG_BIOS_VMX_OFF = required`
- `trait:CUSTOM_BM_CONFIG_RAID_DISK_STRIPE = required`

Its possible the operator has set all of the Ironic nodes with `COMPUTE_A` as the resource class to have all of these traits assigned:

- `CUSTOM_BM_CONFIG_BIOS_VMX_ON`
- `CUSTOM_BM_CONFIG_BIOS_VMX_OFF`
- `CUSTOM_OTHER_TRAIT_I_AM_USUALLY_IGNORED`
- `CUSTOM_BM_CONFIG_RAID_DISK_MIRROR`
- `CUSTOM_BM_CONFIG_RAID_DISK_STRIPE`

The Operator has also defined the following deploy templates:

```
{
  "deploy-templates": [
    {
      "name": "CUSTOM_BM_CONFIG_RAID_DISK_MIRROR",
      "steps": [
        {
          "interface": "raid",
          "step": "create_configuration",
          "args": {
            "logical_disks": [
              {
                "size_gb": "MAX",
                "raid_level": "1",
```

(continues on next page)

(continued from previous page)

```
                "is_root_volume": true
              }
            ],
            "delete_configuration": true
          },
          "priority": 10
        }
      ]
    },
    {
      "name": "CUSTOM_BM_CONFIG_RAID_DISK_STRIPE",
      "steps": [
        {
          "interface": "raid",
          "step": "create_configuration",
          "args": {
            "logical_disks": [
              {
                "size_gb": "MAX",
                "raid_level": "0",
                "is_root_volume": true
              }
            ],
            "delete_configuration": true
          },
          "priority": 10
        }
      ]
    },
    {
      "name": "CUSTOM_BM_CONFIG_BIOS_VMX_ON",
      "steps": [...]
    },
    {
      "name": "CUSTOM_BM_CONFIG_BIOS_VMX_OFF",
      "steps": [...]
    }
  ]
}
```

When a Nova instance is created with `FlavorVMXMirror`, the required traits for that flavor are set on `node.instance_info['traits']` such that Ironic adds the deploy steps defined in `CUSTOM_BM_CONFIG_BIOS_VMX_ON` and `CUSTOM_BM_CONFIG_RAID_DISK_MIRROR`, and the node is appropriately configured for workloads that want that specific flavor.

Alternatives

Alternative approach

This design solves two problems:

1. I want to request some custom configuration to be applied to my bare metal server during provisioning.
2. Ensure that my instance is scheduled to a bare metal node that supports the requested configuration.

As with capabilities, the proposed design uses a single field (traits) to encode configuration and scheduling information. An alternative approach could separate these two concerns.

Deploy templates could be requested by a name (not necessarily a trait) or UUID as a nova flavor `extra_spec`, and pushed to a `deploy_templates` field in the ironic nodes `instance_info` field by the nova virt driver. Ironic would then apply the requested deploy templates during provisioning.

If some influence in the scheduling process is required, this could be provided by traits, but this would be a separate concern.

Adapting the earlier example:

`FlavorVMXMirror` has these extra specs:

- `resource:CUSTOM_COMPUTE_A = 1`
- `trait:CUSTOM_BM_CONFIG_BIOS_VMX_ON = required`
- `trait:CUSTOM_BM_CONFIG_RAID_DISK_MIRROR = required`
- `deploy_template:BIOS_VMX_ON=<?>`
- `deploy_template:BIOS_RAID_DISK_MIRROR=<?>`

Only ironic nodes supporting the `CUSTOM_BM_CONFIG_BIOS_VMX_ON` and `CUSTOM_BM_CONFIG_RAID_DISK_MIRROR` traits would be scheduled to, and the nova virt driver would set `instance_info.deploy_templates` to `BIOS_VMX_ON,BIOS_RAID_DISK_MIRROR`.

There are some benefits to this alternative approach:

- It would automatically support cases beyond the simple one trait mapping to one deploy template case we have here. For example, to support deploy template `X`, features `Y` and `Z` must be supported by the node (without combinatorial trait explosions).
- In isolation, the configuration mechanism is conceptually simpler - the flavor specifies a deploy template directly.
- It would work in standalone ironic installs without introducing concepts from placement.
- We dont overload the concept of traits for carrying configuration information.

There are also some drawbacks:

- Additional complexity for users and operators that now need to apply both traits and deploy templates to flavors.
- Less familiar for users of capabilities.
- Having flavors that specify resources, traits and deploy templates in `extra_specs` could leave operators and users scratching their heads.

Extensions

This spec attempts to specify the minimum viable feature that builds on top of the deployment steps framework specification. As such, there are many possible extensions to this concept that are not being included:

- While you can use standard traits as names of the deploy templates, it is likely that many operators will be forced into using custom traits for most of their deploy templates. We could better support the users of standard traits if we added a list of traits associated with each deploy template, in addition to the trait based name. This list of traits will act as an alias for the name of the deploy template, but this alias may also be used by many other deploy templates. The node validate will fail if for any individual node one of traits set maps to multiple deploy templates. To disambiguate which deploy template is requested, you can look at what deploy template names are in the chosen nodes trait list. For each deploy template you look at any other traits that can be used to trigger that template, eventually building up a trait to deploy template mapping for each trait set on the node (some traits will not map to any deploy template). That can be used to detect if any of the traits on the node map to multiple deploy templates, causing the node validate to fail.
- For some operators, they will end up creating a crazy number of flavors to cover all the possible combinations of hardware they want to offer. It is hoped Nova will eventually allow operators to have flavors that list possible traits, and a default set of traits, such that end users can request the specific set of traits they require in addition to the chosen flavor.
- While ironic inspector can be used to ensure each node is given an appropriate set of traits, it feels error prone to add so many traits to each Ironic node. It is hoped when a concept of node groups is added, traits could be applied to a group of nodes instead of only applying traits to individual nodes (possibly in a similar way to host aggregates in Nova). One suggestion was to use the Resource Class as a possible grouping, but that is only a very small part of the more general issue of groups nodes to physical networks, routed network segments, power distribution groups, all mapping to different ironic conductors, etc.
- There were discussions about automatically detecting which Deploy Templates each of the nodes supported. However most operators will want to control what is available to only the things they wish to support.

Data model impact

Two new database tables will be added for deploy templates:

```
CREATE TABLE deploy_templates (
    id INT(11) NOT NULL AUTO_INCREMENT,
    name VARCHAR(255) CHARACTER SET utf8 NOT NULL,
    uuid varchar(36) DEFAULT NULL,
    PRIMARY KEY (id),
    UNIQUE KEY `uniq_deploy_templaes0uuid` (`uuid`),
    UNIQUE KEY `uniq_deploy_templaes0name` (`name`),
)

CREATE TABLE deploy_template_steps (
    deploy_template_id INT(11) NOT NULL,
    interface VARCHAR(255) NOT NULL,
    step VARCHAR(255) NOT NULL,
    args TEXT NOT NULL,
```

(continues on next page)

(continued from previous page)

```
    priority INT NOT NULL,
    KEY `deploy_template_id` (`deploy_template_id`),
    KEY `deploy_template_steps_interface_idx` (`interface`),
    KEY `deploy_template_steps_step_idx` (`step`),
    CONSTRAINT `deploy_template_steps_ibfk_1` FOREIGN KEY (`deploy_template_
↪id`) REFERENCES `deploy_templates` (`id`),
)
```

The `deploy_template_steps.args` column is a JSON-encoded object of step arguments, `JsonEncodedDict`.

New `ironic.objects.deploy_template.DeployTemplate` and `ironic.objects.deploy_template_step.DeployTemplateStep` objects will be added to the object model. The deploy template object will provide support for looking up a list of deploy templates that match any of a list of trait names.

State Machine Impact

No impact beyond that already specified in the deploy steps specification.

REST API impact

A new REST API endpoint will be added for deploy templates, hidden behind a new API microversion. The endpoint will support standard CRUD operations.

In the following API, a UUID or trait name is accepted for a deploy templates identity.

List all

List all deploy templates:

```
GET /v1/deploy-templates
```

Request: empty

Response:

```
{
  "deploy-templates": [
    {
      "name": "CUSTOM_BM_CONFIG_RAID_DISK_MIRROR",
      "steps": [
        {
          "interface": "raid",
          "step": "create_configuration",
          "args": {
            "logical_disks": [
              {
                "size_gb": "MAX",
                "raid_level": "1",
                "is_root_volume": true
              }
```

(continues on next page)

(continued from previous page)

```
            ],
            "delete_configuration": true
          },
          "priority": 10
        }
      ],
      "uuid": "8221f906-208b-44a5-b575-f8e8a59c4a84"
    },
    {
      ...
    }
  ]
}
```

Response codes: 200, 400

Policy: admin or observer.

Show one

Show a single deploy template:

```
GET /v1/deploy-templates/<deploy template ident>
```

Request: empty

Response:

```
{
  "name": "CUSTOM_BM_CONFIG_RAID_DISK_MIRROR",
  "steps": [
    {
      "interface": "raid",
      "step": "create_configuration",
      "args": {
        "logical_disks": [
          {
            "size_gb": "MAX",
            "raid_level": "1",
            "is_root_volume": true
          }
        ],
        "delete_configuration": true
      },
      "priority": 10
    }
  ],
  "uuid": "8221f906-208b-44a5-b575-f8e8a59c4a84"
}
```

Response codes: 200, 400, 404

Policy: admin or observer.

Create

Create a deploy template:

```
POST /v1/deploy-templates
```

Request:

```
{
  "name": "CUSTOM_BM_CONFIG_RAID_DISK_MIRROR",
  "steps": [
    {
      "interface": "raid",
      "step": "create_configuration",
      "args": {
        "logical_disks": [
          {
            "size_gb": "MAX",
            "raid_level": "1",
            "is_root_volume": true
          }
        ],
        "delete_configuration": true
      },
      "priority": 10
    }
  ],
}
```

Response: as for show one.

Response codes: 201, 400, 409

Policy: admin.

Update

Update a deploy template:

```
PATCH /v1/deploy-templates/{deploy template ident}
```

Request:

```
[
  {
    "op": "replace",
    "path": "/name"
    "value": "CUSTOM_BM_CONFIG_RAID_DISK_MIRROR"
  },
  {
    "op": "replace",
```

(continues on next page)

(continued from previous page)

```
    "path": "/steps"
    "value": [
      {
        "interface": "raid",
        "step": "create_configuration",
        "args": {
          "logical_disks": [
            {
              "size_gb": "MAX",
              "raid_level": "1",
              "is_root_volume": true
            }
          ],
          "delete_configuration": true
        },
        "priority": 10
      }
    ]
  }
]
```

Response: as for show one.

Response codes: 200, 400, 404, 409

Policy: admin.

The `name` and `steps` fields can be updated. The `uuid` field cannot.

Delete

Delete a deploy template:

```
DELETE /v1/deploy-templates/{deploy template ident}
```

Request: empty

Response: empty

Response codes: 204, 400, 404

Policy: admin.

Client (CLI) impact

ironic CLI

None

openstack baremetal CLI

In each of the following commands, a UUID or trait name is accepted for the deploy templates identity.

For the `--steps` argument, either a path to a file containing the JSON data or `-` is required. If `-` is passed, the JSON data will be read from standard input.

List deploy templates:

```
openstack baremetal deploy template list
```

Show a single deploy template:

```
openstack baremetal deploy template show <deploy template ident>
```

Create a deploy template:

```
openstack baremetal deploy template create --name <trait> --steps <deploy␣
↪steps>
```

Update a deploy template:

```
openstack baremetal deploy template set <deploy template ident> [--name
↪<trait] [--steps <deploy steps>]
```

Delete a deploy template:

```
openstack baremetal deploy template delete <deploy template ident>
```

In these commands, `<deploy steps>` are in JSON format and support the same input methods as clean steps - string, file or standard input.

RPC API impact

None

Driver API impact

None

Nova driver impact

Existing traits integration is enough, only now the selected traits on boot become more important.

Ramdisk impact

None

Security impact

Allowing the deployment process to be customised via deploy templates could open up security holes. These risks are mitigated, as seen through the following observations:

- Only admins can define the set of allowed traits for each node.

- Only admins can define the set of requested traits for each Nova flavor, and allow access to that flavor for other users.
- Only admins can create or update deploy templates via the API.
- Deploy steps referenced in deploy templates are defined in driver code.

Other end user impact

Users will need to be able to discover what each Nova flavor does in terms of deployment customisation. Beyond checking requested traits and cross-referencing with the ironic deploy templates API, this is deemed to be out of scope. Operators should provide sufficient documentation about the properties of each flavor. The ability to look up a deploy template by trait name should help here.

Scalability impact

Increased activity during deployment could have a negative impact on the scalability of ironic.

Performance Impact

Increased activity during deployment could have a negative impact on the performance of ironic, including increasing the time required to provision a node.

Other deployer impact

Deployers will need to ensure that Nova flavors have required traits set appropriately.

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
Mark Goddard (mgoddard)

Other contributors:

- Dmitry Tantsur (dtantsur)
- Ruby Loo (rloo)

Work Items

- Add DB tables and objects for deploy templates
- Write code to map traits to deploy templates
- Extend node validation to check all deploy templates are valid
- Add API to add deploy templates
- Extend CLI to support above API
- Write tests

Dependencies

- Node traits spec http://specs.openstack.org/openstack/ironic-specs/specs/approved/node-traits.html
- Deploy steps spec http://specs.openstack.org/openstack/ironic-specs/specs/11.1/deployment-steps-framework.html

Testing

Unit tests will be added to ironic. Tempest API tests will exercise the deploy templates CRUD API.

Upgrades and Backwards Compatibility

The deploy steps API endpoint will be hidden behind a new API version.

During normal operation when the ironic conductor is not pinned, deploy templates will be used to add deploy steps during node provisioning, even if the caller of the node state API uses a microversion that does not support deploy templates.

During an upgrade when the ironic conductor is pinned, deploy templates will not be used to add deploy steps during node provisioning.

Documentation Impact

- Admin guide on how to configure Nova flavors and deploy templates
- Update API ref
- Update CLI docs

References

- http://specs.openstack.org/openstack/ironic-specs/specs/11.1/deployment-steps-framework.html
- http://specs.openstack.org/openstack/ironic-specs/specs/approved/node-traits.html

5.15.125 Deployment Steps Framework

https://storyboard.openstack.org/#!/story/1753128

There is a desire for ironic to support customizable and extendable deployment steps, which would provide the ability to prepare bare metal nodes (servers) that better match the needs of the users who will be using the nodes.

In order to support that, we propose refactoring the existing deployment code in ironic into a deployment steps framework, similar to the cleaning steps framework.

Problem description

Presently, ironic provides a way to prepare nodes prior to them being made available for deployment (see state diagram). This is done via cleaning. However, it is not always possible, efficient, or effective to perform some of these preparations without knowing the requirements of the users of the nodes. In addition, there may be operations that should only be done once the users requirements are known.

For example, during cleaning, a node could be configured for RAID. However, this might not be the desired RAID configuration that the user of the node wants. Since the users desires are only known at

deployment time, a mechanism that allows for custom RAID configuration during deployment is preferred.

Features like custom RAID configuration, BIOS configuration, and custom kernel boot parameters are a few use cases that would benefit from a way of defining deployment steps at deploy time, in ironic.

It makes sense to provide support for this via deployment steps. This would be conceptually similar to the cleaning steps supported by ironic already.

Proposed change

This proposal is the first step in providing support for performing different deployment operations based on the users desires. (The RFE to reconfigure nodes on deploy using traits is an example of a feature that depends on this work.)

The proposed change is to implement a deployment steps (or `deploy steps`) framework that is very similar to the existing framework for automated and manual cleaning. (This was discussed and agreed upon in principle, at the OpenStack Dublin PTG.)

This change is internal to ironic. Users will not be able to affect the deployment process any more than they can do today.

Conceptually, the clean steps model is a simple idea and operators are familiar with it. Having similar deploy steps provides consistency and it will be easier for operators to adopt, due to their familiarity with clean steps. It is also powerful in that, at the end of the day (or year or two), a particular step could be a clean step, a deploy step, or both.

This includes re-factoring of code to be used by both clean and deploy steps.

The existing deployment process will be implemented as a list of one (or more) deploy steps.

What is a deploy step?

Similar to clean steps, functions that are deploy steps will be decorated with `@deploy_step`, defined in ironic/drivers/base.py as follows:

```
def deploy_step(priority, argsinfo=None):
   """Decorator for deployment steps.

   :param priority: an integer priority; used for determining the order in
       which the step is run in the deployment process. (See below,
       "When are deploy steps executed" for more details.)
   :param argsinfo: a dictionary of keyword arguments where key is the name of
       the argument and value is a dictionary as follows:

           description: <description>. Required. This should include
                        possible values.
           required: Boolean. Optional; default is False. True if this
                     argument is required.
```

An alternative is to have one decorator that allows specifying a function to be a clean step and/or a deploy step, e.g.:

```
@step(clean_priority=0, deploy_priority=0, argsinfo=None)
```

However, clean steps are abortable and deploy steps arent (yet, see below), and it is unclear whether other arguments might be added for the deploy step decorator. Thus, it seems safer and simpler to have a separate decorator for deploy steps. (Having one decorator for both types of steps is left as a future exercise.)

Although ironic allows cleaning to be aborted, ironic doesnt allow the deployment to be aborted (although there is an RFE to support abort in deploy_wait). So it is outside the scope of this specification.

A deploy step can be implemented by any Interface, not just DeployInterface.

When are deploy steps executed?

Each deploy step has a priority; a non-negative integer. In this first phase, the priorities will be hard-coded. There will be no way to turn off or change these priorities.

The steps are executed from highest priority to lowest priority. Steps with priorities of zero (0) are not executed. A step has to be finished, before the next one is started.

Alternatives

There may be other ways to provide support for customizable deployment steps per user/instance, but there doesnt seem to be good reasons for having a different design from that used for clean steps.

We could choose not to provide support for customized deploy steps on a per user/instance basis. In that case, some of the current workarounds to overcome this problem include:

- have groups of nodes configured in advance (using clean steps) for each required combination of configurations. This could lead to strange capacity planning issues.
- executing the desired configuration steps after each node is deployed. As these configuration steps are executed post-deploy, most of them need a reboot of the node, orchestration is needed to do these reboots properly, and this causes performance issues that are not acceptable in a production environment. This approach wont work for pre-deploy steps though, such as RAID for the boot disk.
- users can create their own images for each use case. But the limitation is that the number of images can grow exponentially, and that there is no ability to match a specific type of hardware with a specific image.
- use a customizable DeployInterface like the ansible deploy interface (although the ansible deploy interface is not recommended for production use). This may not be able to achieve the same level of access to the hardware or settings, to have the same effect.

Data model impact

Similar to clean steps, a Node object will be updated with:

- a new `deploy_step` field: this is the current deploy step that is being executed or None if no steps have been executed yet. This will require an update to the DB.
- `driver_internal_info['deploy_steps']`: the list of deploy steps to be executed.
- `driver_internal_info['deploy_step_index']`: the index into the list of deploy steps (or None if no steps have been executed yet); this corresponds to node.deploy_step.

State Machine Impact

No new state or transition will be added.

The state of the node will alternate from states.DEPLOYING (`deploying`) to states.DEPLOYWAIT (`wait call-back`) for each asynchronous deploy step.

REST API impact

There will not be any new API methods.

GET /v1/nodes/*

The GET /v1/nodes/* requests that return information about nodes will be modified to also return the nodes `deploy_step` field and the deploy-related information in the nodes `driver_internal_info` field.

Similar to the `clean_step` field, the `deploy_step` field will be the current deploy step being executed, or None if there is no deployment in progress (or hasnt started yet).

If the deployment fails, the `deploy_step` field will show which step caused the deployment to fail.

This change requires a new API version. For nodes that have not yet been deployed using the deploy steps, the `deploy_step` field will be None, and there wont be any deploy-related entries in the `driver_internal_info` field.

For older API versions, this `deploy_step` field will not be available, although any deploy-related entries in the `driver_internal_info` field will be shown.

Client (CLI) impact

The only change (when the new API version is specified), is that the response for a Node will include the new `deploy_step` field and during deployment, the new deploy-step-related entries in the nodes `driver_internal_info` field.

ironic CLI

Even though this has been deprecated, responses will include the change described above.

openstack baremetal CLI

Responses will include the change described above.

RPC API impact

None.

Driver API impact

Similar to cleaning, these methods will be added to the drivers.base.BaseInterface class:

```
def get_deploy_steps(self, task):
    """Get a list of deploy steps this interface can perform on a node.

```

(continues on next page)

(continued from previous page)

```
    :param task: a TaskManager object, useful for interfaces overriding this␣
↪method
    :returns: a list of deploy step dictionaries
    """

def execute_deploy_step(self, task, step):
    """Execute the deploy step on task.node.

    :param task: a TaskManager object
    :param step: The dictionary representing the step to execute
    :raises DeployStepFailed: if the step fails
    :returns: None if this method has completed synchronously, or
        states.DEPLOYWAIT if the step will continue to execute
        asynchronously.
    """
```

The actual deploy steps will be determined in the coding phase; we will start with one big deploy step (to get the framework in) and then break that step up into more steps determined by what makes sense given the existing code, and the constraints (e.g. support for out-of-tree drivers, backwards compatibility when a deploy step in release N is split into several steps in release N+1).

(This specification will be updated with the actual deploy steps, once that is determined.)

Out-of-tree Interfaces

Although the conductor will still support deployment the old way (without deploy steps), this support will be deprecated and removed based on the standard deprecation policy. (The deprecation period may be extended if there is a strong desire to do so by the vendors; were flexible.)

For out-of-tree interfaces that dont have deploy steps, the conductor will emit (log) a deprecation warning, that the out-of-tree interface should be updated to use deploy steps, and that all nodes that are being deployed using the old way, need to be finished deploying, before an upgrade to the release where there is no longer any more support for the old way.

Nova driver impact

None

Ramdisk impact

There should be no impact to the ramdisk (IPA).

In the future, when we allow configuration and specification of deploy steps per node, we might provide support for collecting deploy steps from the ramdisk, but that is out of scope for this first phase.

Security impact

None

Other end user impact

None.

Scalability impact

None.

Performance Impact

None.

Other deployer impact

None.

Developer impact

DeployInterfaces (and any other interfaces involved in the deployment process) will need to be written with deploy steps in mind.

Implementation

Assignee(s)

Primary assignee:

- rloo (Ruby Loo)

Work Items

Ironic:

- Add support for deploy steps to base driver
- rework the existing code into one or more deploy steps
- Update the conductor to get the deploy steps and execute them

`python-ironicclient`:

- Add support for node.deploy_step

Dependencies

None.

Testing

- unit tests for all new code and changed behaviour
- CI jobs already test the deployment process; they should continue to work with these changes

Upgrades and Backwards Compatibility

- Old Interfaces will work with the new BaseInterface class because the code will cleanly fall back when an Interface does not support `get_deploy_steps()`. A deprecation warning will be logged, and we will remove support for the old way according to the OpenStack policy for deprecations & removals.
- Likewise, an Interface implementation with `get_deploy_steps()` will work in an older version of Ironic.
- In a cold upgrade:
  - if the agent heartbeats and driver_internal_info[deploy_steps] is empty, proceed the old way.
  - if a deployment is started by a conductor using deploy steps (new code), it means all the conductors are using the new code, so the deployment can continue on any conductor that supports the node
- In a rolling upgrade:
  - if the agent heartbeats and driver_internal_info[deploy_steps] is empty, proceed the old way (similar to cold upgrade)
  - a new conductor will not use the deploy steps mechanism if it is pinned to the old release (via *pin_release_version* configuration option). if a deployment is started by a conductor using deploy steps (new code), it means that it is unpinned, and all the conductors are using the new code, so the deployment can continue on any conductor that supports the node.

Documentation Impact

- api-ref: https://developer.openstack.org/api-ref/baremetal/ will be updated to include the new node.deploy_step field

References

- cleaning
- OpenStack Dublin PTG etherpad
- RFE to reconfigure nodes on deploy using traits
- RFE to support abort in deploy_wait
- state diagram

5.15.126 Dell EMC hardware firmware update

https://storyboard.openstack.org/#!/story/2003494

Operators and deployers (such as tripleo) need the ability to flash the firmware on Dell EMC hardware to specific versions before provisioning baremetal servers.

Problem description

Use cases

- An operator upgrades the firmware for certain hardware components on a server to newer versions to take advantage of new features or bug fixes in the firmware. This could be prior to or after provisioning.

- An operator rolls back the firmware for certain hardware components on a server to prior versions to avoid regressions introduced in newer firmware. This could be prior to or after provisioning.
- A deployer (such as tripleo) pins the firmware for certain hardware components to specified versions prior to initiating overcloud deployment.

The following use cases are considered outside the scope of this spec:

- An operator or software component uploads a firmware image to a firmware image repository.
- An operator or software component removes a firmware image from a firmware image repository.

Dell EMC WSMAN firmware management

WSMAN firmware management is offered by the iDRAC. Supported operations are:

- List firmware on server: Enumerate DCIM_SoftwareIdentity
- Update firmware: Invoke DCIM_SoftwareInstallationService.InstallFromSoftwareIdentity

Dell EMC Redfish firmware management

Redfish firmware management is offered by the iDRAC. Supported operations are:

- List firmware on server: https://\protect\T1\textdollaridrac_ip/redfish/v1/UpdateService/FirmwareInventory
- Update firmware: https://\protect\T1\textdollaridrac_ip/redfish/v1/UpdateService/Actions/UpdateService.SimpleUpdate

Support for firmware management in `redfish` hardware type

A patch for firmware update in sushy has been merged: https://review.opendev.org/#/c/613828/ There is no support for firmware update in the `redfish` hardware type yet.

Proposed change

Since the Dell EMC redfish implementation of firmware update is fully compliant with the DMTF spec, the decision has been made to go with a `redfish` implementation.

A manual clean step will be added to the RedfishManagement class to initiate firmware update. The clean step will accept a list of dictionaries. Each dictionary will represent a single firmware update, and will contain a URI to the firmware image.

As an example:

```
"clean_steps": [{
    "interface": "management",
    "step": "update_firmware",
    "args": {
        "firmware_images":[
            {
                "url": "file:///firmware_images/idrac/9/iDRAC-with-Lifecycle-
↪Controller_Firmware_VRYKT_WN64_3.32.32.32_A00.EXE",
                "checksum": "<sha1-checksum-of-this-file>"
            },
```

(continues on next page)

(continued from previous page)

```
            {
                "url": "swift://firmware_container/BIOS_W8Y0W_WN64_2.1.7.EXE",
                "checksum": "<sha1-checksum-of-this-file>"
            }
        ]
    }
}]
```

The implementation will apply the firmware updates in the given order.

The implementation will have no knowledge of dependencies of the supplied firmware, or if the firmware is applicable to the hardware that it is being installed on. The implementation will rely on the firmware update failing gracefully in these cases.

The updater will fail fast so that if one update fails, it will abort and not apply the remaining updates. If a failure does occur midway through applying the updates, successful updates prior to the failed update will not be rolled back.

The cleaning step will be out-of-band. The firmware update cleaning step will use Redfish to perform the update. The intent is to use the sushy library if possible, and if not, provide vendor extensions as necessary.

While the iDRAC supports rolling back to the last known good firmware, the ability to do this will not be implemented as part of this spec. Instead, if a user wishes to roll back to an early version of the firmware, they will just do a firmware update to an older version.

While the initial implementation of this will use the Redfish protocol, it will be implemented in such a way that it will not preclude adding support for the `WSMAN` protocol at a later date.

Alternatives

One alternative would be to implement firmware update using `WSMAN`. Because `WSMAN` will eventually be deprecated in favor of Redfish, it is preferred to avoid this option.

Another alternative would be to do firmware update in-band via an Ironic Python Agent hardware manager for the iDRAC.

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

ironic CLI

Users will be able to launch a cleaning step to update the firmware on Dell EMC servers.

openstack baremetal CLI

Users will be able to launch a cleaning step to update the firmware on Dell EMC servers.

RPC API impact

None

Driver API impact

A cleaning step will be added to update the firmware on Dell EMC hardware managed by the `redfish` hardware type.

Nova driver impact

None

Ramdisk impact

None

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

To use firmware update, the user will need to configure the selected hardware type to use the `redfish` management interface and set `redfish` credentials in the nodes driver_info.

Config options

A new [firmware_update] group will be defined in the ironic configuration file. The following options will be moved from the iLO section to that group.

use_web_server_for_images
Indicates if images should be uploaded to the conductor web server.

swift_container
The swift container for firmware images.

swift_object_expiry_timeout
The timeout in seconds after which the given swift URL should expire.

Developer impact

None

Implementation

Assignee(s)

cdearborn

Work Items

- Ironic RedfishManagement changes to add a cleaning step
- python-dracclient changes to implement firmware update
- Ironic iDRAC hardware type changes to add support for the Redfish management interface

Dependencies

None

Testing

Addition of unit tests to test the firmware update cleaning step.

Upgrades and Backwards Compatibility

If a firmware update is attempted on a Dell EMC server that does not support the Redfish UpdateService.SimpleUpdate firmware upgrade command, then cleaning will be aborted and an appropriate error message logged.

Documentation Impact

The documentation will be updated to cover this new feature. The documentation will be updated to include the generations of Dell EMC hardware officially supported.

References

- Manual cleaning - https://github.com/openstack/ironic-specs/blob/master/specs/approved/manual-cleaning.rst
- sushy - https://github.com/openstack/sushy

5.15.127 Driver composition reform

https://bugs.launchpad.net/ironic/+bug/1524745

This spec suggests revamping how we name and compose drivers from interfaces. It will allow having one vendor driver with options configurable per node instead of many drivers for every vendor.

Problem description

Our driver interface matrix has become increasingly complex. To top it all off nowadays we have many interfaces that can be used for every driver. To name a few:

- `boot`: most drivers support PXE and iPXE, while some also support virtual media; support for *petitboot* bootloader is proposed.
- `deploy`: two deploy approaches are supported: write image via iSCSI or write it directly from within the agent.
- `inspect`: there is generic inspection using ironic-inspector, but some drivers also allow out-of-band inspection. This feature is optional, so we should provide a way to disable it.

Currently weve ended up with a complex and really confusing naming scheme. For example:

- `pxe_ipmitool` uses PXE or iPXE for boot and iSCSI for deploy.
- `agent_ipmitool` actually also uses PXE or iPXE, but it does not use iSCSI.
- To top it all, `pxe_ipmitool` is actually using agent!
- To reflect all the possibilities, the names would have to be more like `pxe_iscsi_ipmitool`, `ipxe_iscsi_ipmitool`, `pxe_direct_ipmitool`, `ipxe_direct_ipmitool`, etc.
- Now repeat the same with every power driver we have.

Proposed change

Introduction

The following concepts are used in this spec:

vendor
: The driving force behind a specific driver. It can be hardware vendors or the ironic team itself in case of generic drivers, such as IPMI. This also includes out-of-tree drivers.

hardware interface (or just interface)
: A notion replacing the term driver interface - a set of functionality dealing with some aspect of bare metal provisioning in a vendor-specific way. For example, right now we have `power`, `deploy`, `inspect`, `boot` and a few more interfaces.

hardware type
: A family of hardware supporting the same set of interfaces from the ironic standpoint. This can be as wide as all hardware supporting the IPMI protocol or as narrow as several hardware models supporting some specific interfaces.

driver
: A thin object containing links to hardware interfaces. Before this spec *driver* meant roughly the same as *hardware type* means in this spec.

classic driver
An ironic driver from before this spec: a class with links to interfaces hardcoded in the Python code.

dynamic driver
A *driver* created at run time with links to interfaces generated based on information in the node record (including hardware type and interfaces).

With this spec we are going to achieve the following goals:

- Make *vendors* in charge of defining a set of supported interface implementations in priority order.
- Allow *vendors* to guarantee that unsupported interface implementations will not be used with hardware types they define. This is done by having a hardware type list all interfaces it supports.
- Allow 3rd parties to create out-of-tree *hardware types* that allow them to maximize their reuse of the in-tree interface implementations.
- Make *hardware type* definition as declarative as possible.
- Allow a user to switch *hardware type* for a node, just as it was possible to change a driver before this spec.
- Allow a user to switch between *interface* implementations supported by a *hardware type* for a node via the bare metal API.

Configuration

- A *hardware type* is defined as a Python class - see *Hardware Types* for details. An entry point is created to provide a simple name for each *hardware type*, for example:

```
ironic.hardware.types =
    generic-ipmi = ironic.hardware.ipmi:GenericIpmiHardware
    ilo-gen8 = ironic.hardware.ilo:iLOGen8Hardware
    ilo-gen9 = ironic.hardware.ilo:iLOGen9Hardware
```

- The list of *hardware interfaces* is still hardcoded in the Python code and cannot be extended by plugins. The interfaces are implemented in the same way as before this spec: by subclassing an appropriate abstract class from ironic.drivers.base.
- For each *hardware interface*, all implementations get their own entrypoint and a unique name, for example:

```
ironic.hardware.interfaces.power =
    ipmitool = ironic.drivers.modules.ipmitool:IpmitoolPower
```

- Compatibility between *hardware types* and *hardware interface* implementations is expressed in the Python code - see *Hardware Types* for details.
- Create a new configuration option `enabled_hardware_types` with a list of enabled *hardware types*. This will not include *classic drivers* which are enabled by the existing `enabled_drivers` option.
- Create a family of configuration options `default_<INTERFACE>_interface` that allows an operator to explicitly set a default interface for new nodes upon creation, if one is not specified in the creation request.

  Here `<INTERFACE>` is a type of interface: power, management, etc.

- Create a family of configuration options `enabled_<INTERFACE>_interfaces` with a list of enabled implementations of each *hardware interface* that are available for use in the ironic deployment.

  The default value, if provided by the `default_<INTERFACE>_interface`, must be in this list, otherwise a conductor will fail to start.

- If no interface implementation is explicitly requested by a user in a node creation request, use the value calculated as follows:

  - If `default_<INTERFACE>_interface` is set, use its value. Return an error if it is not supported by the *hardware type* of the node.
  - Otherwise choose the first available interface implementation from an intersection of the `enabled_<INTERFACE>_interfaces` as defined in the deployments configuration and the *hardware_type*s priority ordered list of supported_<INTERFACE>_interfaces. Return an error, if this intersection is empty.

  This calculated default will be stored in the database entry for the node upon creation.

- Change how we load drivers instead of one singleton instance of a driver, well have an instance of *dynamic driver* per node, containing links to hardware interface implementations (just like today).

  However, interface implementations themselves will stay singletons, and will be preloaded during the start up and stored in the conductor.

  Conductor will fail to start if any **enabled** *hardware types* or *interface* implementations cannot be loaded (e.g. due to missing dependencies).

  > **Note**
  >
  > While its technically possible to enable interfaces that are not used in any of enabled *hardware types*, they will not get loaded in this case.

  The *classic drivers* will be loaded exactly as before.

- Modify the periodic tasks collection code to also collect periodic tasks for enabled interfaces of every enabled *hardware type*.

- Conductor will fail to start if there is a name clash between a *classic driver* and a *hardware type*.

Database and Rest API

- Allow the node `driver` field to accept the *hardware types* as well. This will work in all API versions.

  > **Note**
  >
  > There are two reasons for that:
  >
  > - Consistency: we never prevented new drivers to be used with old API versions, and *dynamic drivers* will look mostly like new drivers to users.
  > - Usability: we plan on eventually deprecating the classic drivers. When we remove them, all clients will need to specify the *hardware types* when enrolling nodes. To allow older

clients to continue interacting with the API service, even as they use new driver names (hardware types), we must continue to use the same field name and API semantics.

- For each interface create a new field on the `node` table named `<interface_name>_interface`. A migration will be needed each time we add a new interface (which hopefully wont happen too often).

  For *hardware types* setting `<interface_name>_interface` field to `None` means using the calculated default value as described in *Configuration*.

  Trying to set any of these fields to a value other than `None` will result in an error if the `driver` field is set to a *classic driver*. Similarly, all these fields are reset to `None` if the `driver` field is set to a *classic driver*.

- Every time `driver` and/or any of the interface fields is updated, the conductor checks that the *hardware type* supports all the resulting interfaces (except when `driver` is set to a *classic driver*).

  To change between two incompatible sets of interfaces, all changes should come in one API call. E.g. for a node with the `ilo-gen8` *hardware type* and `vmedia_ilo` boot interface the following JSON patch will be allowed:

```
[
    {"op": "replace", "path": "/boot_interface", "value": "ipxe"},
    {"op": "replace", "path": "/driver", "value": "generic-ipmi"}
]
```

  but the following patch will fail because of incompatible boot interface:

```
[
    {"op": "replace", "path": "/driver", "value": "generic-ipmi"},
]
```

  Note

  RFC 6902 requires a JSON patch to be atomic, because an HTTP PATCH operation must be atomic. Meaning, its possible for some operations to end up with an inconsistent object as long as the end result is consistent.

  The validation will be conducted on the API service side by checking the new `conductor_hardware_interfaces` database table.

- If for some reason the existing *interface* becomes invalid for a node (e.g. it was disabled after the node was enrolled), it will be signalized via the usual node validation API. The validation for this interface wont pass with an appropriate error message. On the programming level, the driver attribute for this interface (e.g. `task.driver.deploy`) will be set to `None`.

- Update `GET /v1/drivers` to also list enabled *hardware types*. This change is **not** affected by API versioning, because we allow old API versions to use *hardware types* with the `driver` field.

- Allow `GET /v1/drivers` to filter only *hardware types* or only *classic drivers*.

  Update `GET /v1/drivers/<HW TYPE>` to report the *hardware type* information, including the list of enabled *hardware interfaces*.

This feature is guarded by an API version bump (as usual).

- Allow filtering nodes by `<interface_name>_interface` fields in the node list API.

  This feature is guarded by an API version bump (as usual).

- Create a new table `conductor_hardware_interfaces` to hold the relationship between conductors, hardware types and available interfaces. A warning will be issued on conductor start up, if it detects that other conductors have a different set of interfaces for the same enabled *hardware type*. This will also track the default interface for each hardware type and interface type combination.

  This situation is inevitable during live upgrades, so it must not result in an error. However, we will document that all conductors should have the same set of interfaces for the same enabled *hardware types*.

  This table will not be exposed in the HTTP API for now.

Deprecations

We are **not** planning to deprecate and remove the support for *classic drivers* in the V1 API.

We are planning to deprecate and remove the *classic drivers* which exist in-tree. The deprecation procedure may be tricky and will be covered by a follow-up spec.

Alternatives

- We could put interfaces under a new JSON key on a node. However, were trying to move away from informally defined JSON keys. It would also prevent us from being able to implement the filtering of nodes based on a particular interface.
- We could create a new API endpoint for updating the interfaces. This will be inconsistent with how we update the `driver` field though.

  We could then create a new API version, preventing updating `driver` via the regular node update API, but that would be a breaking change.

- We could create a new field `hardware_type` instead of having the existing `driver` field accept a *hardware type*. This was a part of the proposal previously, but we found that it complicates things substantially without clear benefits.
- We could create a whole new family of API endpoints instead of reusing `/v1/drivers`, e.g. `/v1/hardware-types`. However, it would require us to replicate all driver-related functionality nearly intact, for example driver vendor passthru. So users would have to somehow figure out which vendor passthru endpoint to use based on what kind of a driver is in the `driver` field.

Data model impact

- For each interface, create a new node field `<interface_name>_interface` initially set to `NULL`.
- Create a new internal table `conductor_hardware_interfaces`:

  `conductor_id` - conductor ID (foreign key to conductors table),

  `hardware_type VARCHAR(255)` - *hardware type* entrypoint name,

  `interface_type VARCHAR(16)` - interface type name (e.g. `deploy`),

  `interface_name VARCHAR(255)` - interface implementation entry point name.

default TINYINT(1) - boolean which denotes if this interface_name is
the default for a given hardware_type and interface_type combination.

This table will get populated on conductor start up and purged on deleting the conductor record. On conductor startup, during init_host(), the conductor will fetch the list of hardware interfaces supported by all registered conductors and compare to its own configuration. If the same *hardware type* is enabled on two conductors with a different set of enabled_interfaces, this will result in a WARNING log message. The enabled *hardware types* themselves do not have to match (just like today, different conductors can have different set of drivers).

State Machine Impact

None

REST API impact

- Update GET /v1/drivers:

  Return both *classic drivers* and *hardware types* no matter which API version is used.

  New URL parameters:

  - type (string, one of classic, dynamic, optional) - if provided, limit the resulting driver list to only *classic drivers* or *hardware types* accordingly.

  New response field:

  type whether the driver is *dynamic* or *classic*.

  This change is guarded by a new API version.

- Update GET /v1/drivers/<NAME>:

  New response field:

  type whether the driver is *dynamic* or *classic*.

  New response fields that are not None only for *hardware types*:

  default_<interface_name>_interface
  the entrypoint name of the calculated default implementation for a given interface.

  enabled_<interface_name>_interfaces
  the list of entrypoint names of enabled implementations for a given interface.

- Update GET /v1/drivers/<NAME>/properties and GET /v1/drivers/<NAME>/vendor_passthru/methods and the actual driver vendor passthru call implementation:

  When requested for a *dynamic driver*, assume the calculated defaults for the vendor interface implementation as described in *Configuration*. We will need to support non-default implementations as well, but it goes somewhat beyond the scope of this already big spec.

Client (CLI) impact

ironic CLI

- Update the node creation command to accept one argument per interface. Example:

```
ironic node-create --driver=ilo-gen9 --power-interface=redfish
```

The same change is applied to the OSC plugin.

- Extend the output of the `driver-list` command with the `Type` column.
- Extend the `driver-list` command with `--type` argument, which, if supplied, limits the driver list to only *classic drivers* (`classic` value) or *hardware types* (`dynamic` value).
- Extend the output of the `driver-show` command with the newly introduced fields.

openstack baremetal CLI

Similar changes to whats in *ironic CLI* are applied here.

RPC API impact

- No impact on the hash ring, as both *hardware types* and *classic drivers* are used in the same field.

Driver API impact

Hardware Types

- Create a new `AbstractHardwareType` class as an abstract base class for all hardware types. Here is a simplified example implementation, using only power, deploy and inspect interfaces:

```
import abc, six

@six.add_metaclass(abc.ABCMeta)
class AbstractHardwareType(object):
    @abc.abstractproperty
    def supported_power_interfaces(self):
        pass

    @abc.abstractproperty
    def supported_deploy_interfaces(self):
        pass

    @property
    def supported_inspect_interfaces(self):
        return [NoopInspect]
```

Note that some interfaces (power, deploy) are mandatory, while the other (inspect) are not. A dummy implementation will be provided for all optional interfaces. Depending on the specific call it will either do nothing or raise an error. For user-initiated calls (e.g. start inspection), an error will be returned. For internal calls (e.g. attach cleaning ports), no action will be taked.

- Create a new `GenericHardwareType` class which most of the actual hardware type classes will want to subclass. This class will insert generic implementations for some interfaces:

```
class GenericHardwareType(AbstractHardwareType):
    supported_deploy_interfaces = [AgentDeploy]
    supported_inspect_interfaces = [NoopInspect, InspectorInspect]
```

Note that all properties contain classes, not instances. Also note that order matters: in this example `NoopInspect` will be the default, if both implementations are enabled in the configuration.

- Here is an example of how hardware types could be created:

```
class GenericIpmiHardware(GenericHardwareType):
    supported_power_interfaces = [IpmitoolPower, IpminativePower]

class iLOGen8Hardware(GenericHardwareType):
    supported_power_interfaces = (
        GenericIpmiHardware.supported_power_interfaces
        + [IloPower]
    )
    supported_inspect_interfaces = (
        GenericHardwareType.supported_inspect_interfaces
        + [IloInspect]
    )

class iLOGen9Hardware(iLOGen8Hardware):
    supported_power_interfaces = (
        iLOGen8Hardware.supported_power_interfaces
        + [RedfishPower]
    )
```

Note

These definitions use classes, not entrypoints names. These examples assume the required classes are imported.

Note

The following entrypoints will have to be defined for these examples to work:

```
ironic.hardware.types =
    generic-ipmi = ironic.hardware.ipmi:GenericIpmiHardware
    ilo-gen8 = ironic.hardware.ilo:iLOGen8Hardware
    ilo-gen9 = ironic.hardware.ilo:iLOGen9Hardware

ironic.hardware.interfaces.power =
    ipmitool = ironic.drivers.modules.ipmitool:IpmitoolPower
    ipminative = ironic.drivers.modules.ipmitool:IpminativePower
    ilo = ironic.drivers.modules.ilo:IloPower
    redfish = ironic.drivers.modules.redfish:RedfishPower

ironic.hardware.interfaces.inspect =
    inspector = ironic.drivers.modules.inspector:InspectorInspect
    ilo = ironic.drivers.modules.ilo:IloInspect
```

The following configuration will be required to enable everything in these examples:

```
[DEFAULT]
enabled_hardware_types = generic-ipmi,ilo-gen8,ilo-gen9
enabled_power_interfaces = ipmitool,ipminative,ilo,redfish
enabled_inspect_interfaces = inspector,ilo
```

Driver Creation

- At start up time the conductor instantiates all enabled hardware types, as well as all enabled interface implementations for enabled hardware types.
- Each time the node is created or loaded from the database, a thin BareDriver object is created with all interfaces set on it. This is similar to how network drivers already work. It gets assigned to `task.driver`, and after that everything works as before this spec.

Nova driver impact

None

Ramdisk impact

None

Security impact

None

Other end user impact

- End users should switch to *hardware types* over time.

Scalability impact

None

Performance Impact

- A driver instance will be now created per node as opposed to creating one per conductor right now. This will somewhat increase the memory usage per node. We can probably define __slots__ on the driver class to reduce this effect.

Other deployer impact

- A deployer can set the new `enabled_hardware_types` option to enable more *hardware types*. Otherwise only the default *hardware types* and already enabled classic drivers will be available.
- A deployer can also set any of new `enabled_<INTERFACE>_interfaces` options to enable more *interfaces* for the enabled *hardware types*.

Developer impact

This spec changes the way we expect the developers to write their drivers.

- No more new *classic drivers* will be accepted in-tree as soon as this change lands.
- Developers should implement *hardware types* and *interfaces* to provide new hardware support for Ironic. Built-in *interfaces* implementations will be available for reuse both in-tree and out-of-tree.

Implementation

Assignee(s)

- Dmitry Tantsur (lp: divius, irc: dtantsur)
- Jim Rollenhagen (irc: jroll)

Work Items

- Create base classes supporting *hardware types*.
- Create tables for tracking enabled *hardware interfaces*.
- Load *hardware types* on conductor start up and record them in the internal table.
- Create node fields for *interfaces* and expose them in the API.
- Update the drivers API to support *hardware types*.
- Create the *hardware types* for hardware supported directly by the team, i.e. the generic IPMI-compatible hardware. The SSH driver might be removed soon; it wont get updated in this case.

Dependencies

- For the vendor interface to be really pluggable, we need to promote agent passthru to the core API.

Testing

- Unit test coverage will obviously be provided.
- A new gate job will be created, using a dynamic version of the IPMI driver. We will aim to make it the primary approach in the gate over time.
- Grenade testing for upgrades / migration of existing workloads to new drivers.

Upgrades and Backwards Compatibility

This reform is designed to be backward compatible. The *classic drivers* will be supported for at least some time. A separate spec will cover the deprecation of the *classic drivers*.

We will recommend switching to using appropriate *dynamic drivers* as soon as its possible.

Upgrade flow

1. Ironic is updated to a version supporting *dynamic drivers*. The API version used by clients is not updated yet.
2. All nodes are still using *classic drivers*. On a node `driver=x_y`.
3. Users with an old API version:

- can set `driver` to a *classic driver*.
- can set `driver` to a *hardware type*, which will result in using a *dynamic driver* with the default set of interfaces.

4. Users with a new API version:
   - can set `driver` to a *hardware type* or a *classic driver*
   - can set non-default interface implementations when `driver` is set to a real *hardware type*

Documentation Impact

- Document switching to *dynamic drivers*
- Document creating new *hardware types*

References

Initial etherpad: https://etherpad.openstack.org/p/liberty-ironic-driver-composition

Newton etherpad: https://etherpad.openstack.org/p/ironic-newton-summit-driver-composition

5.15.128 Dynamic port groups

https://bugs.launchpad.net/ironic/+bug/1652630

In the Ocata release, Ironic added support for grouping ports into port groups. The administrator also has the ability to specify `mode` and `properties` of a port group by using appropriate port group fields. However the administrator is still required to pre-create a port group on the ToR (Top of Rack) switch manually. Or find some other way to sync portgroup settings between Ironic and the ML2 driver.

Problem description

While Ironic provides the ability to create port groups with different configurations (mode and properties), port groups still have to be pre-created on ToR manually as we do not pass port group details to ML2 drivers. To make port groups creation `dynamic` Ironic should pass port group settings to Neutron ML2 drivers. The ML2 driver is responsible for port group configuration (creation/deletion) on ToR switch during deployment.

This spec does not cover end-user interface for specifying port group configuration when doing `nova boot` at this moment. It can be extended when community has some agreement on it.

Proposed change

Start passing port group information to Neutron ML2 drivers via the Neutron port `binding:profile` field. Appropriate Neutron ML2 drivers will use that information to create port group dynamically on the ToR switch.

> **Note**
>
> We cannot use existing `binding:profile` `local_link_information` key as it is a list with port details, where `switch_id` and `port_id` are mandatory keys. For portgroup object those keys are not required as portgroup is virtual interface and might be spread across different switches. For example MLAG configuration.

Binding profile data structure

- Introduce new `local_group_information` dictionary that stores portgroups information.
- Reuse existing `local_link_information` for port objects only

A JSON example of `binding:profile` with `local_link_information` reuse:

```
"binding:profile": {
    'local_link_information': [
        {
            'switch_info': 'tor-switch0',
            'port_id': 'Gig0/1'
            'switch_id': 'aa:bb:cc:dd:ee:ff'
        },
        {
            'switch_info': 'tor-switch0',
            'port_id': 'Gig0/2',
            'switch_id': 'aa:bb:cc:dd:ee:ff'
        }
    ],
    'local_group_information': {
            'id': '51a9642b-1414-4bd6-9a92-1320ddc55a63',
            'name': 'PortGroup0',
            'bond_mode': 'active-backup',
            'bond_properties': {
                'bond_xmit_hash_policy': 'layer3+4',
                'bond_miimon': 100,
            }
        },
}
```

The data types:

Field Name	Description
id	The UUID of Ironic portgroup object
name	The name of the ironic port group
bond_mode	Ironic portgroup mode
bond_properties	Ironic portgroup properties
switch_info	The hostname of the switch
port_id	The identifier of the port on the switch
switch_id	The identifier of the switch, ie mac address

Note

It is recommended to pick `bond_mode` and keys/values for `bond_properties` from the[1] as they will be used by user OS.

[1] *Linux kernel bond*: https://www.kernel.org/doc/Documentation/networking/bonding.txt

Alternatives

- Use port groups in the static fashion when administrator pre-creates port group on ToR switch.
- If ML2 driver supports port group creation, make sure that port group properties in Ironic and ML2 are the same.

Data model impact

None.

State Machine Impact

None.

REST API impact

None.

Client (CLI) impact

None.

RPC API impact

None.

Driver API impact

None.

Nova driver impact

None.

Ramdisk impact

None.

Security impact

None.

Other end user impact

None.

Scalability impact

None.

Performance Impact

None.

Other deployer impact

No need to pre-create port group at the ToR switch. Only need to specify port group configuration at the Ironic portgroup object.

Developer impact

Out of tree network interfaces should be updated to pass `portgroup.mode` and `portgroup.properties` with `links` array in Neutron port `binding:profile` field. Vendors are responsible to deal with `links` to support dynamic port groups.

Implementation

Assignee(s)

Primary assignee:
vsaienko <vsaienko@mirantis.com>

Work Items

- Update `neutron` network interface to pass data structure described in *Binding profile data structure* to Neutron.
- Add dynamic port group support to networking-generic-switch
- Update tempest with appropriate tests.

Dependencies

Dynamic portgroup support is dependent on Neutron ML2 driver functionality being developed to deal with `links` array in `binding:profile` field.

Testing

- Add dynamic port group support to networking-generic-switch
- Update tempest with appropriate tests.

Upgrades and Backwards Compatibility

Backward compatibility is retained as Ironic will still pass `local_link_information` in Neutron port `binding:profile` field.

Documentation Impact

This feature will be fully documented.

References

5.15.129 Enhance Power Interface for Soft Power Off and NMI

https://bugs.launchpad.net/ironic/+bug/1526226

The proposal presents the work required to enhance the power interface to support soft reboot and soft power off, and the management interface to support diagnostic interrupt (NMI [1]).

Problem description

There exists a problem in the current driver interface which doesnt provide with soft power off and diagnostic interrupt (NMI [1]) capabilities even though ipmitool [2] and most of BMCs support these capabilities.

Here is a part of ipmitool man page in which describes soft power off and diagnostic interrupt (NMI [1]).

$ man ipmitool:

```
...
power

       Performs a chassis control command to view and change the
       power state.

       ...

       diag

              Pulse a diagnostic interrupt (NMI) directly to the
              processor(s).

       soft

              Initiate a soft-shutdown of OS via ACPI. This can be
              done in a number of ways, commonly by simulating an
              overtemperature or by simulating a power button press.
              It is necessary for there to be Operating System
              support for ACPI and some sort of daemon watching for
              events for this soft power to work.
```

From customers point of view, both tenant admin and tenant user, the lack of the soft power off and diagnostic interrupt (NMI [1]) lead the following inconveniences.

1. Customer cannot safely shutdown or soft power off their instance without logging on.
2. Customer cannot take NMI dump to investigate OS related problem by themselves.

From deployers point of view, that is cloud provider, the lack of the two capabilities leads the following inconveniences.

1. Cloud provider support staff cannot shutdown customers instance safely without logging on for hardware maintenance reason or etc.
2. Cloud provider support staff cannot ask customer to take NMI dump as one of investigation materials.

Proposed change

In order to solve the problems described in the previous section, this spec proposes to enhance the power states, the PowerInterface base class and the ManagementInterface base class so that each driver can implement to initiate soft reboot, soft power off and inject NMI.

And this enhancement enables the soft reboot, soft power off and inject NMI through Ironic CLI and REST API for tenant admin and cloud provider. Also this enhancement enables them through Nova CLI and REST API for tenant user when Novas blueprint [3] is implemented.

As a reference implementation, this spec also proposes to implement the enhanced PowerInterface base class into the IPMIPower concrete class and the enhanced ManagementInterface base class into the IPMIManagement concrete class.

1. add the following new power states to ironic.common.states:

```
SOFT_REBOOT = 'soft rebooting'
SOFT_POWER_OFF = 'soft power off'
```

2. add get_supported_power_states method and its default implementation to the base PowerInterface class in ironic/drivers/base.py:

```
def get_supported_power_states(self, task):
    """Get a list of the supported power states.

    :param task: A TaskManager instance containing the node to act on.
    :returns: A list of the supported power states defined
              in :mod:`ironic.common.states`.
    """
    return [states.POWER_ON, states.POWER_OFF, states.REBOOT]
```

   - Note: WakeOnLanPower driver supports only states.POWER_ON.

3. add a default parameter `timeout` into the set_power_state method in to the base PowerInterface class in ironic/drivers/base.py:

```
@abc.abstractmethod
def set_power_state(self, task, power_state, timeout=None):
    """Set the power state of the task's node.

    :param task: a TaskManager instance containing the node to act on.
    :param power_state: Any power state from :mod:`ironic.common.states`.
    :param timeout: timeout positive integer (> 0) for any power state.
       ``None`` indicates to use default timeout which depends on
       ``power_state``[*]_ and driver.
    :raises: MissingParameterValue if a required parameter is missing.
    """
```

4. enhance set_power_state method in IPMIPower class so that the new states can be accepted as power_state parameter.

   IPMIPower reference implementation supports SOFT_REBOOT and SOFT_POWER_OFF.

   SOFT_REBOOT is implemented by first SOFT_POWER_OFF and then a plain POWER_ON such that Ironic implemented REBOOT. This implementation enables generic BMC detect the reboot

completion as the power state change from ON -> OFF -> ON which power transition is called `power cycle`.

The following table shows power state value of each state variables. `new_state` is a value of the second parameter of set_power_state() function. `power_state` is a value of node property. `target_power_state` is a value of node property.

new_state	power_state (start state)	target_power_state (assigned value)	power_state (end state)
SOFT_REBOOT	POWER_ON	SOFT_POWER_OFF	POWER_OFF[*]_
SOFT_REBOOT	POWER_OFF[*]_	POWER_ON	POWER_ON
SOFT_POWER_OFF	POWER_OFF	POWER_ON	POWER_ON
SOFT_POWER_OFF	POWER_ON	SOFT_POWER_OFF	POWER_OFF
	POWER_OFF	NONE	POWER_OFF

new_state	power_state (start state)	target_power_state (assigned value)	power_state (end state)
SOFT_REBOOT	POWER_ON	SOFT_POWER_OFF	ERROR[*]_
SOFT_POWER_OFF	POWER_ON	SOFT_POWER_OFF	ERROR[*]_

5. add get_supported_power_states method and implementation in IPMIPower:

```
def get_supported_power_states(self, task):
    """Get a list of the supported power states.

    :param task: A TaskManager instance containing the node to act on.
       currently not used.
    :returns: A list of the supported power states defined
              in :mod:`ironic.common.states`.
    """

    return [states.POWER_ON, states.POWER_OFF, states.REBOOT,
            states.SOFT_REBOOT, states.SOFT_POWER_OFF]
```

6. add inject_nmi abstract method to the base ManagementInterface class in ironic/drivers/base.py:

```
@abc.abstractmethod
def inject_nmi(self, task):
    """Inject NMI, Non Maskable Interrupt.

    :param task: A TaskManager instance containing the node to act on.
    :returns: None
    """
```

7. add inject_nmi concrete method implementation in IPMIManagement class.

Alternatives

- Both the soft power off and diagnostic interrupt (NMI [1]) could be implemented by vendor passthru. However the proposed change is better than the vendor passthru, because users of Ironic API or Ironic CLI can write script or program uniformly.

Data model impact

None

State Machine Impact

None

REST API impact

- Add support of SOFT_REBOOT and SOFT_POWER_OFF to the target parameter of following API:

```
PUT /v1/nodes/(node_ident)/states/power

The target parameter supports the following JSON data respectively.
``timeout`` is an optional parameter for any ``target`` parameter.
In case of "soft reboot" and "soft power off", ``timeout`` overrides
``soft_power_off_timeout`` in the in the Ironic configuration file,
typically /etc/ironic/ironic.conf.

Examples

  {"target": "soft reboot",
   "timeout": 900}

  {"target": "soft power off",
   "timeout": 600}
```

- Add a new management API to support inject NMI:

```
PUT /v1/nodes/(node_ident)/management/inject_nmi

Request doesn't take any parameter.
```

Client (CLI) impact

- Enhance Ironic CLI ironic node-set-power-state to support power graceful off/reboot by adding optional arguments. This CLI is async. In order to get the latest status, call ironic node-show-states and check the returned value.:

```
usage: ironic node-set-power-state <node> <power-state>
       [--soft] [--timeout <timeout>]

Power a node on/off/reboot, power graceful off/reboot to a node.
```

(continues on next page)

(continued from previous page)

```
Positional arguments

<node>

    Name or UUID of the node.

<power-state>

    'on', 'off', 'reboot'

Optional arguments:
   --soft
     power graceful off/reboot.

   --timeout <timeout>
     timeout positive integer value(> 0) for any ``power-state``.
     If ``--soft`` option is also specified, it overrides
     ``soft_power_off_timeout`` in the in the Ironic configuration
     file, typically /etc/ironic/ironic.conf.
```

- Add a new Ironic CLI ironic node-inject-nmi to support inject nmi. This CLI is async. In order to get the latest status, serial console access is required.:

```
usage: ironic node-inject-nmi <node>

Inject NMI, Non Maskable Interrupt.

Positional arguments

<node>

    Name or UUID of the node.
```

- Enhance OSC plugin openstack baremetal node so that the parameter can accept reboot [soft] [timeout <timeout>], power [on|off [soft] [timeout <timeout>] and inject nmi. This CLI is async. In order to get the latest status, call openstack baremetal node show and check the returned value.:

```
usage: openstack baremetal node reboot [--soft] [--timeout <timeout>]
↪<uuid>

usage: openstack baremetal node power off [--soft] [--timeout <timeout>]
↪<uuid>

usage: openstack baremetal node inject nmi <uuid>
```

RPC API impact

None

Driver API impact

PowerInterface base and ManagementInterface base are enhanced by adding a new method respectively as described in the section Proposed change. And these enhancements keep API backward compatible. Therefore it doesnt have any risk to break out of tree drivers.

Nova driver impact

The default behavior of nova reboot command to a virtual machine instance such as KVM is soft reboot. And nova reboot command has a option hard to indicate hard reboot.

However the default behavior of nova reboot to an Ironic instance is hard reboot, and hard option is meaningless to the Ironic instance.

Therefore Ironic Nova driver needs to be update to unify the behavior between virtual machine instance and bare-metal instance.

This problem is reported as a bug [6]. How to fix this problem is specified in nova blueprint [10] and spec [11].

The default behavior change of nova reboot command is made by following the standard deprecation policy [12]. How to deprecate nova command is also specified in nova blueprint [10] and spec [11].

Ramdisk impact

None

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

- Deployer, cloud provider, needs to set up ACPI [7] and NMI [1] capable bare metal servers in cloud environment.
- change the default timeout value (sec) in the Ironic configuration file, typically /etc/ironic/ironic.conf if necessary.

Developer impact

- Each driver developer needs to follow this interface to implement this proposed feature.

Implementation

Assignee(s)

Primary assignee:
Naohiro Tamura (naohirot)

Other contributors:
None

Work Items

- Enhance PowerInterface class and ManagementInterface class to support soft power off and inject nmi [1] as described Proposed change.
- Enhance Ironic API as described in REST API impact.
- Enhance Ironic CLI as described in Client (CLI) impact.
- Implement the enhanced PowerInterface class into the concrete class IPMIPower, and the enhanced ManagementInterface class into the concrete class IPMIManagement. Implementing vendors concrete class is up to each vendor.
- Coordinate the work with Nova NMI support Inject NMI to an instance [3] if necessary.
- Update the deployer documentation from the ironic perspective.

Dependencies

- Soft power off control depends on ACPI [7]. In case of Linux system, acpid [8] has to be installed. In case of Windows system, local security policy has to be set as described in Shutdown: Allow system to be shut down without having to log on [9].
- NMI [1] reaction depends on Kernel Crash Dump Configuration. How to set up the kernel dump can be found for Linux system in [13], [14], and for Windows in [15].

Testing

- Unit Tests.
- Tempest Tests, at least soft reboot/soft power off.
- Each vendor plans Third Party CI Tests if implemented.

Upgrades and Backwards Compatibility

None (Forwards Compatibility is out of scope)

- Note The backwards compatibility issue of the default behavior change of nova reboot command is solved by following the standard deprecation policy [12].

Documentation Impact

- The deployer doc and REST API reference manual need to be updated. (CLI manual is generated automatically from source code)

References

[1] http://en.wikipedia.org/wiki/Non-maskable_interrupt

[2] http://linux.die.net/man/1/ipmitool

[3] https://review.opendev.org/#/c/187176/

[4] https://en.wikipedia.org/wiki/Communicating_sequential_processes

[5] http://linux.die.net/man/1/virsh

[6] https://bugs.launchpad.net/nova/+bug/1485416

[7] http://en.wikipedia.org/wiki/Advanced_Configuration_and_Power_Interface

[8] http://linux.die.net/man/8/acpid

[9] https://technet.microsoft.com/en-us/library/jj852274%28v=ws.10%29.aspx

[10] https://blueprints.launchpad.net/nova/+spec/soft-reboot-poweroff

[11] https://review.opendev.org/#/c/229282/

[12] http://governance.openstack.org/reference/tags/assert_follows-standard-deprecation.html

[13] https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/html/Kernel_Crash_Dump_Guide/

[14] https://help.ubuntu.com/lts/serverguide/kernel-crash-dump.html

[15] https://support.microsoft.com/en-us/kb/927069

5.15.130 API Evolution: etag ID

As time has gone on, the API surface has evolved, and the community has given rise to more use-cases and more users. This has resulted in the identification of several shortcomings in the current REST API: it is not as user-friendly as one might expect, and it does not service all of the use-cases we wish that it would.

This specification describes the implementation of etag identifiers on API responses, allowing for better conflict resolution between concurrent API clients.

https://bugs.launchpad.net/ironic/+bug/1605728

Problem description

Multiple clients attempting to update the same resource can overwrite each others changes accidentally and without notice. This is referred to as the lost update problem, and it is a problem in the Bare Metal service. The lost update problem is commonly addressed through the use of etag identifiers, as described in this API Working Group specification on etags. This specification proposes that the Bare Metal service provides etags support to address this problem.

Proposed change

The Bare Metal service shall begin to store, internally, a unique etag identifier for every API modifiable resource (Node, Port, Portgroup, Chassis). This identifier shall be returned in the body and headers of successful responses to GET requests for requested resource beginning with the introduction of a new microversion. There are mainly two cases:

- GET one resource(subresource). The ETag will be returned in the response headers and body. Python clients may operate with resource using fresh etag (see client section). Ironic shell users see etag in the response.
- GET list of resources(subresources). Etag of each resource will be available in response body, as field of the resource:

  {

  ports: [

  {

  uuid: 11111111-2222-3333-4444-555555555555,

  <all other fields>,

  etag: W/eeeeeeeeeeeee

  },

  {

  uuid: 66666666-7777-8888-9999-222222222222,

  <all other fields>,

  etag: W/ttttttttttttt

  }

  ]

  }

  Note that, putting etags to headers does not make sense, because it is impossible to distinguish etags at client side in standard and simple way.

All requests to modify a resource or subresource, whether through a PUT, PATCH, or DELETE request, SHOULD begin to require If-Match header with an appropriate etag identifier to be supplied in the request (for subresource it is etag of that subresource). Note that If-Match will not be required according to RFC standard specification of If-Match Header.

Therefore SHOULD keyword used here which means that `If-Match` header will be optional parameter in client (see keywords to indicate requirement levels).

This is important that originally according to rfc entity-tags using `If-Match` MUST be provided by clients to pass request successfully. When ETags are used for cache validation it is useful. Specifically to our use case making `If-Match` header necessary will decrease user-friendliness of ironic client. It is up to users to decide if they want to be aware of what they change.

`If-None-Match` or any other Precondition Header Fields will not be supported.

To save efficiency of the request the If-Match header SHALL be validated at API and conductor side by comparing the supplied If-Match header against the current etag string. If the supplied header does not match the actual value, a `412 Precondition Failed` SHALL be returned.

On the successful receipt of any request to modify a resource or subresource, a new etag identifier SHALL be generated by the server (it MUST NOT depend on which `ironic-api-version` came in the request). Where the modification is synchronous and the response already includes a representation of the resource, the new etag identifier SHALL be included in the response body and header.

Etag is a SHA-512 hash generated from JSON string encoded compactly (ordering of dict is not needed) from the dictionary of oslo versioned object fields. Etag will be generated for ALL create and modify requests taking into account the fields except of the ignored ones:

For node ignored fields are:

`driver_internal_info, etag, updated_at`

For port, portgroup and chassis it is only `etag` and `updated_at`.

The generated etag will contain W/ prefix to specify that weak validator is used. Strong validators are more useful for comparison, but in our use-case, taking into account, that etag is not changed on every update and not when metadata changes (e.g. Content-Type), weak validators are applied. See weak vs strong validators.

Alternatives

If we do not implement etag support, we will not have any means to prevent races between clients PATCH requests, and we will not be able to implement support for PUT as a means to update resources or sub-resources.

Data model impact

As on every REST API request the objects are obtained from database and while it is not going to be changed, etag shall be retrieved from object returned from db. This is more efficiently than spending time to generate hash again and again.

For that purpose a new field `etag` SHALL be added to each resource table to store the etag identifier. Etag identifier will be String Field with data length limited to 130 characters (etag is a string containing prefix W/ + SHA-512 hexadecimal number 128 digits long).

New field `etag` SHALL be also added to Object model in order to be consistent with db layer. Object layer will also be the place where etag will be regenerated based on current Object fields.

Etag will be also included to notifications payload to make them more flexible and usable.

State Machine Impact

None

REST API impact

A new etag header MUST be sent in response to all GET and POST requests starting from specific API microversion, as well as synchronous PATCH and PUT requests.

The same If-Match etag header SHOULD be accepted in all PUT, PATCH and DELETE requests. That means that each endpoint offering any update capability should have logic to validate etag optioanlly.

A new error status `412 PreconditionFailed` SHALL be introduced and used to signal the clients that their version of a resource is stale, when the supplied etag header does not match the servers version.

Client (CLI) impact

Using new microversion clients get the ability to update resource being aware of what exactly they change. For that they SHOULD send an `If-Match` header with an etag identifier and supported Ironic API version in the header of requests to modify any resource. There are two options: doing this either through CLI or through Python Client API. The last option is available for any python developer script used in clouds (useful for production).

The workflow of etags usage in ironicclient shell:

- Client does GET request.
- Starting from specific API microversion, response SHALL include an etag for requested resource(s) in the headers. ETag SHALL be also included within returned resources in the body, both for GET individual resource and resource collection.
- Etag may be specified by users if needed when doing any operation leading to resource changing by adding `--etag` flag to the command. Etag can be obtained from body or headers of response:

  ```
  ironic --ironic-api-version 1.40 node-update \
  --etag <etag_string> driver_info/foo value
  ```

  This etag string is put as `If-Match` header to the request.
- Ironic API pops `If-Match` header, checks it with rpc_nodes etag and if they match, the entity tag is sent further to RPC where conductor validates it again. If etags are not matching at some point, the `412 PreconditionFailed` error will be raised. If requested `X-Openstack-Ironic-Api-Version` does not support etags yet, `NotAcceptable` error is raised.

To make Python Client API usable without shell, resources will be stored as full-featured objects (not just the bag of attributes), including the etag identifier. To do this the ironicclient API will be rewritten in the way that the `Resource` class is able to update itself and call manager to send requests available in NodeManager. The `Resource` will be stored in the memory like all Python objects are stored during process execution.

For the Python API all appropriate actions of `Resource` object will accept optional parameter etag. The workflow can be the following:

- In Python Shell or in some script clients do GET request to the resource. The etag returned in the response will be stored in the resource representation. E.g.

  `node = node_manager.get(node_ident)`
- Afterwards at any time user scripts can do the action on the resource itself:

  `new_node = node.update(patch, etag=True)`

  Afterwards they will have the up-to-date resource representation if the request will be validated on the server.

  Note, that for 1 standard deprecation period `If-Match` will not be sent to server by default. Clients will be warned that in the next release `etag` parameter will be `True` by default.

If etag is requested it will be retrieved from current resource representation

(as `node.etag` or `getattr(node, 'etag')`). Afterwards it will be sent as `If-Match` header, it means that user cares about up-to-date information. If etag is not present at the resource and clients did not turn off `etag` option, they will fail if using the API version greater or equal than etag API version.

Depending on the situation, the client may choose to transparently retry, or display a diff to the user of the stored vs. server-side resource. Clients should also begin alerting the user when an update request fails due to a resource conflict.

ironic CLI

See above.

openstack baremetal CLI

Same workflow described in the Client Impact.

RPC API impact

RPC API version needs to be bumped to accept etag parameter for actions on the resource. The `etag` parameter, default as None, should be passed to the appropriate methods.

Driver API impact

None

Nova driver impact

Nova ironic driver may use new Ironic API microversion, so ironic api version used by nova virt driver needs to be bumped. Until etag option in python ironicclient API is True, in the nova driver we should explicitly specify `etag=False` in appropriate methods through `Node` resource object.

Ramdisk impact

None

Security impact

None

Other end user impact

Sending `If-Match` header may fail due to `412 Precondition Failed` error. A client may retry with new fresh etag or/and display a diff between two resources representations.

Scalability impact

None

Performance Impact

New etags generation may increase the time to respond depending on the resource size.

Other deployer impact

Some services (e.g. Nova) change baremetal resources through API, so they may upgrade Ironic API to use etag. If services do not upgrade, warn the deployer about that in the case skipping these upgrades may violate some strong recommendations and information consistency is not guaranteed on ironic side.

Developer impact

Python developers can work with `Resource` object as with full-featured objects and perform modifying operations on them. Also they can implement scripts that are using etag option in parallel in efficient way.

Implementation

Assignee(s)

Primary assignee:
galyna

Other contributors:
vdrok

Work Items

- Implement database migration, adding an internal etag field to all top-level resources.
- Implement generation and validation utility functions in common code.
- Implement changes within the ironic.api.controllers.v1 modules to accept and return etag identifiers when fetching or changing resources as appropriate.
- Implement unit tests and tempest tests.
- Update api-ref documentation.
- Implement changes in the python client library and openstack CLI to begin caching etags on GET requests and sending etags on PUT/PATCH/DELETE requests.

Dependencies

None

Testing

Unit and tempest tests shall be added that ensure etag identifiers are returned, that they are validated by requests to modify resources, and that proper errors are returned when an invalid (or merely not current) etag is supplied.

Upgrades and Backwards Compatibility

Backwards compatibility is retained because etags SHOULD only be returned and required in new microversions.

This change does not include substantive changes to any resource.

Documentation Impact

The proper use of etag identifiers shall be documented in our API reference.

References

- https://tools.ietf.org/html/rfc7232
- http://specs.openstack.org/openstack/api-wg/guidelines/etags.html
- https://etherpad.openstack.org/p/ironic-v2-api

5.15.131 Expose conductor information from API

https://storyboard.openstack.org/#!/story/1724474

The spec proposes to expose conductor information from API, as well as the mapping between conductors and nodes.

Problem description

In deployment with multiple ironic conductors incorporated, nodes are distributed across conductors by hashring, currently we have no mechanism to identify the specific conductor for a given node without database query.

There are several cases we need to locate the host of the conductor for a node, for example:

- A node failed to boot from PXE during node deploy, we need to check the PXE environment of the conductor host.
- Ramdisk logs are stored in the conductor host if the backend is set to local, this requires identifying the exact conductor which serves a node.

Meanwhile we are lacking support from API for information related with conductors, which might be useful for diagnostic. For example:

- How many conductors in this cloud, whether they are alive.
- The conductor group each conductor belongs to.
- Nodes currently mapped to this conductor.

This information also makes it possible to implement features like service monitoring, alarming from the outside of ironic.

Proposed change

Adds an API endpoint `/v1/conductors` to retrieve a list of conductor information, including the hostname, conductor group, and alive flag.

The alive flag is calculated in the same way as how ironic assumes a conductor is not alive, by checking `update_at` and `heartbeat_timeout`.

Adds API endpoint `/v/conductors/{hostname}` to retrieve detailed conductor information, including nodes affiliated with this conductor. The mapping is retrieved from hashring.

Adds a field `conductor` to the Node API object, which indicates the hostname of the conductor currently the node is mapped to.

Alternatives

Admins can directly query the ironic database for a conductor list. To locate the last conductor that served a node, query nodes table for the `conductor_affinity` field, then query `conductors` table for the hostname of the conductor. The `conductor_affinity` is updated in several cases, mainly during node deployment time. There is no alternative to get the real time mapping from the hashring.

Data model impact

None

State Machine Impact

None

REST API impact

API microversion will be bumped.

Add a new `conductor` API object, and three API endpoints:

- GET /v1/conductors
  - Get the list of conductors known by ironic
  - Client with earlier microversion before this feature implemented will receive 404. For a normal request, 200 is returned.
  - Sample response data:

```
{
  "conductors": [
    {
      "hostname": "Compute01",
      "conductor_group": "region1",
      "alive": true,
      "links": [
        {
          "href": "http://127.0.0.1:6385/v1/conductors/Compute01",
          "rel": "self"
        },
        {
          "href": "http://127.0.0.1:6385/conductors/Compute01",
          "rel": "bookmark"
        }
      ]
    },
    {
      "hostname": "Compute03",
      "conductor_group": "region1",
      "alive": true,
      "links": [...]
    }
```

(continues on next page)

(continued from previous page)

```
  ]
}
```

- GET /v1/conductors/{hostname}
  - Get detailed information of a conductor by hostname
  - Client with earlier microversion before this feature implemented will receive 404. For a normal request, 200 is returned.
  - Sample response data:

```
{
  "hostname": "Compute01",
  "conductor_group": "region1",
  "alive": true
  "drivers": ["ipmi"],
  "last_seen": "2018-09-10 19:53:17"
  "links": [
    {
      "href": "http://127.0.0.1:6385/v1/conductors/Compute01",
      "rel": "self"
    },
    {
      "href": "http://127.0.0.1:6385/conductors/Compute01",
      "rel": "bookmark"
    }
  ]
}
```

- GET /v1/nodes/{node_ident}/conductor
  - Get detailed information of a conductor for given node
  - Client with earlier microversion before this feature implemented will receive 404. For a normal request, 200 is returned.
  - The response data is same as /v1/conductors/{hostname}

Change `Node` API object in the following way:

- Add a read-only attribute `conductor` to indicate the hostname of associated conductor.
- Retrieve and assign the conductor hostname in `Node.convert_with_links`.

The hostname of the conductor will be returned in these endpoints:

- `POST /v1/nodes`
- `GET /v1/nodes` (when `detail` is set to true)
- `GET /v1/nodes/detail`
- `GET /v1/nodes/{node_ident}`

Sample response data for `GET /v1/nodes/{node_ident}` would be:

```
{
  "nodes": [
    {
      "instance_uuid": null,
      "conductor": "Compute01",
      "uuid": "a308bca6-e6a3-4349-b8ea-695e17672898",
      "links": ...,
      "maintenance": false,
      "provision_state": "available",
      "power_state": "power off",
      "name": "node-0"
    }
  ]
}
```

Add support for querying nodes by conductor hostname for endpoints:

- `GET /v1/nodes`
- `GET /v1/nodes/detail`

For example, `GET /v1/nodes/?conductor=Compute01` will return nodes mapped to the conductor whose hostname is Compute01.

Client (CLI) impact

ironic CLI

None

openstack baremetal CLI

Enhance ironic client with two new command:

- `openstack baremetal conductor list`
- `openstack baremetal conductor show <hostname>`

Expose the conductor field in command:

- `openstack baremetal node list --detail`
- `openstack baremetal node show <node>`

Support node querying by conductor hostname:

- `openstack baremetal node list --conductor <hostname>`

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

None

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

None

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
kaifeng <kaifeng.w@gmail.com>

Work Items

- Implement the API endpoints.
- Add `conductor` field to node API object.
- CLI enhancement for API changes.
- Documentation and api reference.

Dependencies

None

Testing

The feature will be covered by unit tests.

Upgrades and Backwards Compatibility

The feature will be guarded by microversion.

Documentation Impact

New APIs will be documented in the API reference. New commands will be documented in the ironic-client documentation.

References

None

5.15.132 Indicator management

https://storyboard.openstack.org/#!/story/2005342

This spec proposed creating of API for hardware indicators management.

Problem description

Depending on the hardware vendor, bare metal machines might carry on-board and expose through the baseboard management controller (BMC) various indicators, most commonly LEDs.

For example, a blade system might have LEDs on the chassis, on the blades (compute cards), on the drives, on the PSUs, on the NICs.

The use-cases when ironic-manageable LEDs might make sense include:

- The DC staff member want to identify hardware unit in the rack by lighting up its LED in ironic
- The Deployer wants to identify ironic node by pressing a button on the physical unit to light up the LED
- The remote Admin user wants to be aware of the failure LEDs lighting on the bare metal nodes possibly indicating a failure

Proposed change

Overview

This RFE proposes an extension of the `node` ReST API endpoint that will allow reading and toggling the indicators (e.g. LEDs) on the hardware units.

Indicator management workflow

1. An API client can choose to discover available node indicators by sending `GET /v1/nodes/<node_ident>/management/indicators` request. This step is optional.

2. An API client reads the indicator on the chosen component of the bare metal node by sending `GET /v1/nodes/<node_ident>/management/indicators/<component>/<ind_ident>` request and presenting current indicator state to the user.
3. The API client can change the state of the indicator(s) on the given component by sending `PUT /v1/nodes/<node_ident>/management/indicators/<component>` request.

Alternatives

The user can communicate with BMC independently from ironic for the same purpose. Though ironic node correlation with physical node may be challenging.

Data model impact

None.

Due to the interactive nature of indicator information, it seems that user-indicator link should be as immediate as possible, having it cached by the database can make indicators confusing.

State Machine Impact

None.

Indicators should always work the same regardless of the machine state.

REST API impact

- `GET /v1/nodes/<node_ident>/management/indicators`

  Retrieves bare metal node components. Returns a JSON object listing all available node components.

  Currently known components are: `system`, `chassis` and `drive`. Indicator names are free-form, but hopefully descriptive.

  Error codes:

  - 404 Not Found if node is not found.

  Example response object:

```
{
    "components": [
        {
            "name": "system",
            "links": [
                {
                    "href": "http://127.0.0.1:6385/v1/nodes/Compute0/
                    management/indicators/system",
                    "rel": "self"
                },
                {
                    "href": "http://127.0.0.1:6385/nodes/Compute0/
                    management/indicators/system",
                    "rel": "bookmark"
```

(continues on next page)

(continued from previous page)

```
                }
            ]
        },
        {
            "name": "chassis",
            "links": [
                {
                    "href": "http://127.0.0.1:6385/v1/nodes/Compute0/
                    management/indicators/chassis",
                    "rel": "self"
                },
                {
                    "href": "http://127.0.0.1:6385/nodes/Compute0/
                    management/indicators/chassis",
                    "rel": "bookmark"
                }
            ]
        }
    ]
}
```

- `GET /v1/nodes/<node_ident>/management/indicators/<component>`

  Retrieves indicators for a component. Returns a JSON object listing all available indicators for given hardware component along with their attributes.

  Currently known components are: `system`, `chassis` and `drive`. Indicator names are free-form, but hopefully descriptive.

  Error codes:

  – 404 Not Found if node or component is not found.

  Example response object:

```
{
    "indicators": [
        {
            "name": "power",
            "readonly": true,
            "states": [
                "OFF",
                "ON"
            ],
            "links": [
                {
                    "href": "http://127.0.0.1:6385/v1/nodes/Compute0/
                    management/indicators/system/power",
                    "rel": "self"
                },
                {
                    "href": "http://127.0.0.1:6385/nodes/Compute0/
```

(continues on next page)

(continued from previous page)


```
                    management/indicators/system/power",
                    "rel": "bookmark"
                }
            ]
        },
        {
            "name": "alert",
            "readonly": false,
            "states": [
                "OFF",
                "BLINKING",
                "UNKNOWN"
            ],
            "links": [
                {
                    "href": "http://127.0.0.1:6385/v1/nodes/Compute0/
                    management/indicators/system/alert",
                    "rel": "self"
                },
                {
                    "href": "http://127.0.0.1:6385/nodes/Compute0/
                    management/indicators/system/alert",
                    "rel": "bookmark"
                }
            ]
        },
    ]
}
```


- `GET /v1/nodes/<node_ident>/management/indicators/<component>/<ind_ident>`

  Retrieves indicator state for the component. Returns a JSON object representing current state of the chosen indicator (`ind_ident`) sitting on the `component`.

  The field of the response object is `state`, the value is one of: `OFF`, `ON`, `BLINKING` or `UNKNOWN`.

  Error codes:

  - 404 Not Found if node, component or indicator is not found.

  Example response object:


```
{
  "state": "ON"
}
```


- `PUT /v1/nodes/<node_ident>/management/indicators/<component>/<ind_ident>`

  Set the state of the desired indicators of the component. The endpoint accepts a JSON object. The following field is mandatory:

  - `state` requested indicator state
  - 400 Bad Request if `state` is not an accepted value

- 404 Not Found if node, component or indicator is not found.

Example request object:

```
{
  "state": "ON"
}
```

Client (CLI) impact

ironic CLI

None.

openstack baremetal CLI

The following commands will be created:

```
openstack baremetal node indicator list <node> [component]
openstack baremetal node indicator show <node> <component> indicator
openstack baremetal node indicator set <node> <component> <indicator> --state
↪{ON,OFF,BLINKING}
```

The first command lists all indicators for the specified component, or for all components if specific component is not given.

RPC API impact

The new RPC calls are introduced:

- Listing the indicators

```
def get_supported_indicators(self, context, node_id, component=None):
   """Get node hardware components and their indicators.

   :param context: request context.
   :param node_id: node id or uuid.
   :param component: The hardware component, one of
       :mod:`ironic.common.components` or `None` to return all
       available components.
   :returns: a `dict` holding indicator IDs as keys, indicator properties
       as values. Indicator properties is a `dict` that includes:
       `readonly` bool, `states` list containing zero or more values from
       mod:`ironic.common.indicator_states`.
   """
```

- Reading the indicator

```
def get_indicator_state(self, context, node_id, component, indicator):
    """Get node hardware component indicator state.

    :param context: request context.
```

(continues on next page)

(continued from previous page)

```
    :param node_id: node id or uuid.
    :param component: The hardware component, one of
        :mod:`ironic.common.components`.
    :param indicator: Indicator IDs, as
        reported by `get_supported_indicators`
    :returns: current indicator state. One of the values from
        mod:`ironic.common.indicator_states`.
    """
```

- Setting the indicator

```
def set_indicator_state(self, context, node_id, component,
                        indicator, state):
    """Set node hardware components indicator to the desired state.

    :param context: request context.
    :param node_id: node id or uuid.
    :param component: The hardware component, one of
        :mod:`ironic.common.components`.
    :param indicator: Indicator IDs, as
        reported by `get_supported_indicators`)
    :param state: Indicator state, one of
        mod:`ironic.common.indicator_states`.
    """
```

Driver API impact

Optional indicator API methods is added to *ManagementInterface*:

- Listing the indicators

```
def get_supported_indicators(self, task, component=None):
    """Get a map of the supported indicators (e.g. LEDs).

    :param task: A task from TaskManager.
    :returns: A dictionary of hardware components
        (:mod:`ironic.common.components`) as keys with indicator
        properties as values. Indicator properties is a `dict`
        that includes: `readonly` bool, `states` list containing
        zero or more values from mod:`ironic.common.indicator_states`.
    """
```

- Reading the indicator

```
def get_indicator_state(self, task, component, indicator):
    """Get current state of the indicator of the hardware component.

    :param task: A task from TaskManager.
    :param component: The hardware component, one of
        :mod:`ironic.common.components`.
```

(continues on next page)

(continued from previous page)

```
    :param indicator: Indicator ID (as reported by
        `get_supported_indicators`).
    :returns: current indicator state. One of the values from
        mod:`ironic.common.indicator_states`.
    """
```

- Setting the indicator

```
def set_indicator_state(self, task, component, indicator, state):
    """Set indicator on the hardware component to the desired state.

    :param task: A task from TaskManager.
    :param component: The hardware component, one of
        :mod:`ironic.common.components`.
    :param indicator: Indicator ID (as reported by
        `get_supported_indicators`).
    :state: Desired state of the indicator, one of
        :mod:`ironic.common.indicator_states`.
    """
```

The above methods are implemented for Redfish and IPMI hardware types.

Nova driver impact

None.

Ramdisk impact

None.

Security impact

None.

Other end user impact

The indicators can be made accessible through Horizon or other UI tools.

Scalability impact

None.

Performance Impact

None.

Other deployer impact

None.

Developer impact

None.

Implementation

Assignee(s)

Primary assignee:
<etingof>

Work Items

- Add indicator management methods to ironic management interface
- Add indicator management to ironic ipmi and redfish hardware types
- Add RPC for indicator management
- Add REST API endpoint for indicator management

Dependencies

None.

Testing

- Unit tests and Tempest API will be provided

Upgrades and Backwards Compatibility

This change is fully backward compatible.

Documentation Impact

API reference will be provided.

References

5.15.133 Expose supported power states

https://bugs.launchpad.net/ironic/+bug/1734827

This SPEC proposes adding a new API to expose supported power states of nodes. This API would be helpful to see if a power action is supported by a node because it depends on whether a power interface supports power actions such as soft power off and soft reboot.

Problem description

Ironic supports soft shutdown and soft reboot since Ocata. However, not all power interfaces support these new power actions. A new API would be necessary to see if a power action is supported by a node. This proposed API is similar to the one for getting a nodes supported boot devices, which is described here.

Proposed change

The new APIs will be introduced to get the current power status and the supported power states of nodes. See the REST API impact section and for the details:

- GET /v1/nodes/{node_ident}/states/power
- GET /v1/nodes/{node_ident}/states/power/supported

The new CLI will be also added for the new APIs. See the Client (CLI) impact section and for the details:

- `openstack baremetal node power show [--supported] <node>`

Alternatives

There is another option to only add a new API to get supported power states. There are also a few other options to support the API by the CLI:

- Add a new option to display the supported power states to `openstack baremetal node show`.
- Add a new subcommand to show the supported power states like: `openstack baremetal node supported power show`.

However, the current design is better for consistency with the existing APIs and the CLI for the boot devices.

Data model impact

None

State Machine Impact

None

REST API impact

We will introduce the two APIs. They will be available starting with a new Bare Metal API version.

- Get the current power state of a node:

  GET /v1/nodes/{node_ident}/states/power

  The response is like:

```
{
  "power_state": "power on"
}
```

- Get the supported power states of a node:

  GET /v1/nodes/{node_ident}/states/power/supported

  The response contains supported power states of the node: For example:

```
{
  "supported_power_states": [
    "power on",
    "power off",
```

(continues on next page)

(continued from previous page)

```
    "rebooting",
    "soft rebooting",
    "soft power off"
  ]
}
```

Client (CLI) impact

The `openstack baremetal` CLI will support the new API.

ironic CLI

None

A new feature is no longer added to the ironic CLI.

openstack baremetal CLI

A new subcommand will be added:

```
openstack baremetal node power show [--supported] <node>
```

Without the `--supported` option, this command displays the power state of the specified node. When the `--supported` option is specified, this command displays the supported power states.

RPC API impact

A new RPC API, `get_supported_power_states` will be added. This returns a list of the supported power states of the specified node synchronously.

Driver API impact

None

The driver API `get_suuported_power_states` was already defined in the base power interface. If a power interface doesnt override the method, the default list which contains power on, power off, and reboot is returned.

Nova driver impact

None

The new API is available to see if a requested power action is supported or not. Though it might be helpful for the Nova driver, no change is planned currently.

Ramdisk impact

None

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

None

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
shiina-hironori (irc:hshiina)

Other contributors:
None

Work Items

- Add the new RPC API.
- Add the new Baremetal APIs.
- Add the OSC baremetal subcommand.
- Add a new API test to tempest.
- Add the new APIs to the API reference.

Dependencies

None

Testing

An API test will be added to Tempest.

Upgrades and Backwards Compatibility

None

Documentation Impact

The new APIs will be added to the Baremetal API Reference.

References

This change was originally mentioned in reviewing a SPEC to support soft shutdown.: https://review.opendev.org/#/c/186700/

5.15.134 Firmware interface

https://storyboard.openstack.org/#!/story/2010659

This spec proposes to implement a new hardware interface for automated firmware updates.

Problem description

Some operators would like to make sure that their hardware is using specific versions of firmware on different hardware components (e.g. BIOS, NICs, BMC, etc), main reason being they like their machines in cluster to be homogeneous. When they get a new machine they need to update all components firmware by doing upgrade or downgrade.

Currently in Ironic we have support to update the firmware, the operator can achieve this by invoking manual cleaning on the node by enabling the clean step *update_firmware*; the problem is that it requires knowledge about the clean step name and parameters.

- As an operator I want to install specific versions of firmware in my machines (BIOS, NICs, DPUs, GPUs, BMC) before installing the Operating System.

Proposed change

- After a node is enrolled and the basic hardware information is available, an operator can define a Firmware Update config.
- The work to be done here is similar to what we did for *BIOSInterface* and *RAIDInterface*. A new interface *FirmwareInterface* will be created that allows retrieving current installed firmware version of the hardware components in a node and also update their firmware.
- The information about the current firmware version of each hardware component on the node will be collected out-of-band and should be available after we enroll the node via verify step.
- A new clean step *update* will be created for the *FirmwareInterface*, it will be used to update the firmware of each hardware component on the node.
- A new database table *firmware_information* will be created if it doesnt exist. It will contain the information about the current firmware version of each hardware component on a node, the information is updated in case the clean step *update* is called.
- We intend this to start only on the redfish interface, all others will default to *no-firmware*. If other hardware vendors wish to implement it, they are welcome to.

Note

This spec describes the out of band interface. An in-band interface is planned to be implemented later, it can be called *AgentFirmware*, changes on IPA-level APIs will have to be defined. Other implementations can support going through this interface to execute the necessary in-band steps.

Alternatives

We can use the current *update_firmware* clean step via manual cleaning[0], but the downside is that we dont know which hardware components are upgradable and what their present firmware versions are on the node.

Data model impact

- New object *ironic.objects.firmware.Firmware* will be created.
- A new field *firmware_interface* will be added to the *Node* object.
- A new table *firmware_information* will be created, it will store the firmware information of each hardware component of a Node.
  - Table description:
    - node_id
      - Integer
      - PrimaryKeyConstraint(nodes.id)
    - component
      - String
      - PrimaryKeyConstraint
    - initial_version - stored when we create the entry
      - String
    - current_version
      - String
    - last_version_flashed
      - String
    - created_at
      - DateTime
    - updated_at - when the component was last flashed
      - DateTime

[0] Cleaning Steps - https://docs.openstack.org/ironic/latest/admin/cleaning.html

State Machine Impact

None

REST API impact

A new step is proposed to be implemented on the *FirmwareInterface*:

- *firmware.update*: it will trigger the firmware update for each component that is specified. For example:

```
{
  "target":"clean",
  "clean_steps": [{
    "interface": "firmware",
    "step": "update",
    "requires_ramdisk": true,
    "args": {
      "settings": [
        {
          "component": <name>,
          "url": <value>
        },
        {
          "component": <name>,
          "url": <value>
        }
      ]
    }
  }]
}
```

- A new REST API will be introduced to get the cached Firmware information for a node:

```
GET /v1/nodes/<node_ident>/firmware
```

The operation will return the currently cached settings with the following data schema:

```
[
  {
    "component":"bios",
    "initial_version": "v1.0.0.0 (01.02.2022)",
    "current_version": "v1.2.3.4 (01.02.2023)",
    "last_version_flashed": "v1.2.3.4 (01.02.2023)",
    "created_at": "2023-02-01 09:00:00",
    "updated_at": "2023-03-01 10:00:00"
  },
  {
    "component": "bmc",
    "initial_version": "v1.0.0",
    "current_version": "v1.0.0",
    "last_version_flashed": "",
```

(continues on next page)

(continued from previous page)

```
    "created_at": 2023-02-01 09:00:00",
    "updated_at": ""
  }
]
```

Client (CLI) impact

openstackSDK will be updated

- Retrieve all firmware information about the node:

```
$ openstack baremetal node firmware list <node-uuid>
+----+-----------+-----------------------+-----------------------+-----------
↪----------+---------------------------+---------------------------+
| ID | Component | Initial Version       | Current Version       | Last␣
↪Version Flashed  | created_at                | Updated At                |
+----+-----------+-----------------------+-----------------------+-----------
↪----------+---------------------------+---------------------------+
|  1 | bios      | v1.0.0.0 (01.02.2022) | v1.2.3.4 (01.02.2023) | v1.2.3.4␣
↪(01.02.2023) | 2023-02-01T09:00:00.000000 | 2023-03-01T10:00:00.000000 |
+----+-----------+-----------------------+-----------------------+-----------
↪----------+---------------------------+---------------------------+
|  2 | bmc       | v1.0.0                | v1.0.0                |           ␣
↪           | 2023-02-01T09:00:00.000000 |                           |
+----+-----------+-----------------------+-----------------------+-----------
↪----------+---------------------------+---------------------------+
```

RPC API impact

- None - we already have *do_node_clean*

Driver API impact

A new interface `FirmwareInterface` will be available for drivers to allow them to implement the firmware update. The following methods will be available:

- *update(settings)* - This is the step responsible to update the firmware of the components in the node. The *settings* parameter is a list of dictionaries

```
[{"component": "bmc", "url":"<url_new_bmc_fw>"},
 {"component": "bios", "url":"<url_new_bios_fw>"}]
```

- *cache_firmware_information()* - this method will be called to update the firmware information in the *firmware_information* database table. It will store the Firmware information for a node, or update the information in case the *update* step was called.

Nova driver impact

- None

Ramdisk impact

- Currently there is no impact for ramdisk, because we will be focusing on OOB upgrades, the current interface will be created so it can handle in-band upgrades.

Security impact

- None

Other end user impact

- None

Scalability impact

- None

Performance Impact

- The firmware update may extend the time required for manual cleaning on the nodes.

Other deployer impact

- New config options in *ironic.conf*
  - *enabled_firmware_interfaces*: a list of enabled firmware interfaces.
  - *default_firmware_interface*: default firmware interface to be used.
- Operators can use the new steps as part of manual cleaning tasks.

Developer impact

- Developers may implement the *FirmwareInterface* for respective drivers.

Implementation

Assignee(s)

Primary assignee: * <iurygregory, imelofer@redhat.com or iurygregory@gmail.com>

Other contributors: * <dtantsur, dtantsur@protonmail.com> * <janders, janders@redhat.com>

Work Items

- Add the firmware_interface field in the Node object
- Create the Firmware object
- Create the *FirmwareInterface* structure. Includes the redfish implementation
- Implement *no-firmware* and *fake* for the *FirmwareInterface*

- Create REST API
- Implement OSC baremetal CLI changes

Dependencies

- This feature is targeting only hardware that supports Redfish.

Testing

- Unit tests will be added for the code.
- Tempest tests will be added using fake driver.

Upgrades and Backwards Compatibility

- Raise errors when there is no *FirmwareInterface* support in driver.

Documentation Impact

- New Documentation will be provided on how to use.

References

5.15.135 Switch periodic tasks to the Futurist library

https://bugs.launchpad.net/ironic/+bug/1526277

Futurist is a new Oslo library providing tools for writing asynchronous code. This spec suggests switching our periodic task implementation to Futurist to solve some long-standing problems.

Problem description

The main problem with our current implementation is that we run all periodic tasks in one thread. Any task that blocks for a while would block all other tasks from executing, and it happens pretty often with tasks checking power states via IPMI.

Switch to Futurist will allow executing all tasks in parallel.

Proposed change

- Modify conductor to use Futurist library instead of implementation from oslo incubator.

  Existing worker pool will be reused for periodic tasks. So, existing option `workers_pool_size` will set maximum number of tasks to run in parallel at every moment of time.

- Switch all use cases of `ironic.openstack.common.periodic_task` to Futurist decorators, and drop this module.
- Switch `ironic.drivers.base.driver_periodic_task` to using Futurist decorators internally and deprecate it.

Alternatives

- We could fix existing implementation. Thats not actually easier, as it requires essentially rewriting it.

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

RPC API impact

None

Driver API impact

- Old way of creating driver periodic tasks will be deprecated, drivers should eventually switch to using Futurist decorators.

Nova driver impact

None

Ramdisk impact

N/A

Security impact

None

Other end user impact

None

Scalability impact

- Overall positive impact on scalability expected, as every Ironic conductor will be able (at least theoretically) to manage more IPMI nodes.

Performance Impact

- A periodic tasks performance will no longer affect timing of other periodic tasks.

Other deployer impact

None

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
Dmitry Tantsur (irc: dtantsur, lp: divius)

Work Items

- Change conductor manager to use Futurist
- Modify `driver_periodic_task` to use Futurist internally

Dependencies

The only big dependency is Futurist itself. At the moment of writing it didnt see an official release yet, but is moving pretty fast and all required code already landed in git master.

Testing

Unit tests should already cover this functionality. Specific tests ensuring parallelization will be added.

Upgrades and Backwards Compatibility

None

Documentation Impact

Documentation of driver periodic tasks should be updated to mention Futurist instead of ad-hoc implementation.

References

- Futurist periodic task documentation

5.15.136 Hardware interface for BIOS configuration

https://bugs.launchpad.net/ironic/+bug/1712032

The proposal is intended to create a new hardware interface for BIOS automated configuration and a method to make BIOS configuration available as part of manual cleaning[0].

[0] Manual cleaning - https://github.com/openstack/ironic-specs/blob/master/specs/approved/manual-cleaning.rst

Problem description

- There are several use cases that need to configure BIOS options to enable certain functionality or gain performance optimization on OpenStack baremetal nodes. For example, in order to use SRIOV[1] or DPDK[2] technologies, the Virtualization Technology BIOS option shall be enabled; to achieve deterministic low packet latency in real time scenario, BIOS options related to Power Management, CPU sleep states etc shall be disabled; another good example is console redirection BIOS settings.

Proposed change

- After a node is enrolled and the basic hardware information is available, the operator can define a BIOS configuration.
- The operator will be able to set the BIOS configuration for a specific node using a cleaning step which will be defined on the new `BIOSInterface`. The BIOS settings to be changed will be passed in the form of a JSON dictionary as an argument to the cleaning step. This cleaning step will be manual at the moment, but it may be extended to be an automated cleaning step in the future.
- Similar to the RAIDInterface, this proposes a new hardware interface called `BIOSInterface` which will be used for out-of-band BIOS configuration for hardware types. Calling it BIOSInterface doesnt mean the spec is specific to BIOS systems, but a general interface name for systems such as BIOS, UEFI etc. Please refer to Driver API impact for a list of methods that will be added for this interface.

  Note

  The reason of using BIOS instead of others is that its hard to find a proper name that applies to all the systems and BIOS is agreed to be close enough.

- The credentials of BIOS configuration which reuse the existing credentials in `driver_info` will be validated when going from enroll to manageable.
- A new database table `bios_settings` will be created if not exist and the current BIOS settings will be retrieved and updated for the node as part of entering cleaning step in `_do_node_clean`, this means it will be called for both manual and automated cleaning. After the given BIOS options are successfully applied in baremetal, it will update the cached bios settings with applied changes.
- This interface will not be available with any of the classic drivers. And classic drivers would throw UnsupportedDriverExtension for this interface.
- If there is no BIOS interface for a node (i.e. bios_interface=no-bios), an attempt to change BIOS configuration will result in cleaning step not found error.

Alternatives

- Operator can change the BIOS configuration manually whenever required. But this has to be done for each node which is time-consuming and error-prone.

[1] SRIOV BIOS settings - https://docs.openstack.org/neutron/latest/admin/config-sriov.html#create-virtual-functions-compute

[2] DPDK BIOS settings - http://dpdk.org/doc/guides/linux_gsg/sys_reqs.html

Data model impact

- Unlike the RAID configuration[3], the target BIOS settings will be passed as an argument to cleaning step and not stored in the database.
- The current BIOS config will be cached and will be stored in a separate BIOS table. The following database table and fields will be added:
  - A new table named `bios_settings` will be added with the following fields:
    - node_id
      - Integer
      - PrimaryKeyConstraint
      - ForeignKeyConstraint(nodes.id)
    - name
      - String
      - PrimaryKeyConstraint
    - value
      - String
    - created_at
      - DateTime
    - updated_at
      - DateTime

    It will store the cached BIOS information that was retrieved from the node and will be updated when the BIOS settings are changed. created_at and updated_at fields will be updated accordingly when a new record is added or the existing record is updated.
  - `node.bios_interface` will be added to node table, and it will contain the hardware interface we want to use for BIOS automation.
  - New objects `ironic.objects.bios.BIOSSetting` and `ironic.objects.bios.BIOSSettingList` will be added to object model. The `BIOSSetting` and `BIOSSettingList` fields in the python object model will be populated on-demand.

State Machine Impact

- When going from enroll to manageable, credentials are validated to make sure user has the proper rights to access the BIOS config.

REST API impact

Two new cleaning steps are proposed to be implemented on the `BIOSInterface`:

- bios.factory_reset. It will trigger the BIOS settings factory reset for a given node. For example:

[3] RAID configuration - https://github.com/openstack/ironic-specs/blob/master/specs/approved/ironic-generic-raid-interface.rst

```
{
  "target":"clean",
  "clean_steps": [{
    "interface": "bios",
    "step": "factory_reset"
  }]
}
```

- bios.apply_configuration. It will set the given settings to the BIOS of a given node. For example:

```
{
  "target":"clean",
  "clean_steps": [{
    "interface": "bios",
    "step": "apply_configuration",
    "args": {
      "settings": [
        {
          "name": <name>,
          "value": <value>
        },
        {
          "name": <name>,
          "value": <value>
        }
      ]
    }
  }]
}
```

- A new REST API will be introduced to get the cached BIOS config for a node:

```
GET /v1/nodes/<node_ident>/bios
```

The operation will return the currently cached settings with the following data schema:

```
{
  "bios": [
    {
      "links": [
        {
          "href": "http://127.0.0.1:6385/v1/nodes/<node_ident>/bios/<name>
↪",
          "rel": "self"
        },
        {
          "href": "http://127.0.0.1:6385/nodes/<node_ident>/bios/<name>",
          "rel": "bookmark"
        }
      ],
      "name": <name>,
```

(continues on next page)

(continued from previous page)

```
      "value": <value>
    },
    {
      "links": [
        {
          "href": "http://127.0.0.1:6385/v1/nodes/<node_ident>/bios/<name>
↪",
          "rel": "self"
        },
        {
          "href": "http://127.0.0.1:6385/nodes/<node_ident>/bios/<name>",
          "rel": "bookmark"
        }
      ],
      "name": <name>,
      "value": <value>
    }
  ]
}
```

The API will return HTTP 400 (Bad Request) if driver doesnt support BIOS configuration or 404 (Resource Not Found) if node BIOS has not yet been configured. Otherwise it will return HTTP 200 (OK).

- To get a specified BIOS setting for a node:

```
GET /v1/nodes/<node_ident>/bios/<setting name>
```

The operation will return the specified BIOS setting with the following data schema:

```
{
  "<setting name>":
    {
      "name": <setting name>,
      "value": <value>
    }
}
```

Client (CLI) impact

ironic CLI

The ironic CLI will not be updated.

openstack baremetal CLI

- To retrieve the cached BIOS configuration with node-uuid:

```
$ openstack baremetal node bios setting list <node-uuid>
```

- To show a specified BIOS setting with node-uuid:

```
$ openstack baremetal node bios setting show <node-uuid> <setting-name>
```

- The validation result of BIOS Interface will be returned through the standard validation interface.

RPC API impact

None

Driver API impact

A new `BIOSInterface` will be available for the drivers to allow them to implement BIOS configuration. There will be several new methods and cleaning steps in the interface:

- `do_factory_reset()` - This method is called to reset all the BIOS settings supported by driver to factory default. It will also update the records of `bios_settings` database table to the known defaults once reset action succeeds. It is up to the vendor to decide the BIOS defaults settings that will be set.
- `factory_reset()` - This cleaning step will delegate the actual reset work into the abstract method `do_factory_reset()`.

  The operator can choose to call it as part of manual cleaning steps. The corresponding manual cleaning step will be `bios.factory_reset`.
- `do_apply_configuration(configuration={})` - The driver implementation of this method will take the settings from the configuration dictionary and will apply BIOS configuration on the bare metal. The driver is responsible for doing the corresponding validation before applying the settings, and/or manage failures when setting an invalid BIOS config. Implementation of this method needs to rollback previous settings upon first failure. In the case of needing password to update the BIOS config, it will be taken from the `driver_info` properties. The implementation detail is up to the driver.
- `apply_configuration(configuration={})` - This cleaning step will delegate the actual configuration work into the abstract method `do_apply_configuration(configuration={})`.

  The operator can choose to call it as part of manual cleaning steps. The corresponding manual cleaning step will be `bios.apply_configuration`.
- `cache_bios_settings()` - This method will be called to update BIOS configuration in `bios_settings` database table. It will attempt to get the current BIOS settings and store them in the `bios_settings` database table. It will also update the timestamp fields of created_at and updated_at accordingly. The implementation detail is up to the driver, for example, whether to have a sub method shared by `do_factory_reset`, `do_apply_configuration` and `cache_bios_settings` to retrieve and save bios information in `bios_settings` table.

Nova driver impact

None

Ramdisk impact

None

Security impact

Unprivileged access to the BIOS configuration can expose sensitive BIOS information and configurable BIOS options to attackers, which may lead to disruptive consequence. Its recommended that this kind of ability is only restricted to administrative roles. Changing BIOS settings requires credentials which will reuse the existing credentials in `driver_info` instead of creating new fields.

Other end user impact

None

Scalability impact

None

Performance Impact

BIOS configuration may extend the time required for manual cleaning on the nodes.

Other deployer impact

- Add new config options:
  - `enabled_bios_interfaces`: a list of enabled bios interfaces.
  - `default_bios_interface`: default bios interface to be used.
- Operator can use `bios.apply_configuration` and `bios.factory_reset` as manual cleaning tasks for doing BIOS management.

Developer impact

Developer may implement the `BIOSInterface` for respective drivers.

Implementation

Assignee(s)

Primary assignee:
zshi yroblamo

Work Items

- Add bios interface field in Node object.
- Create database model & api for nodes bios table and operations.
- Create `BIOSInterface` which includes the following items:
  - Add new methods in `BIOSInterface` base driver, such as `do_apply_configuration`, `do_factory_reset` and `cache_bios_settings`.
  - Add new cleaning steps in `BIOSInterface` base driver, such as `apply_configuration`, `factory_reset`.
  - Add caching of BIOS config as part of entering cleaning step in `_do_node_clean`.

- Create fake & no-bios implementation derived from `BIOSInterface`.
- Create REST API endpoints for BIOS configuration.
- Create RPC objects for BIOS configuration.
- Implement OSC baremetal CLI changes.

Dependencies

None

Testing

- Unit tests will be added for the code. A fake implementation of the `BIOSInterface` will be provided with `do_apply_configuration` method for testing purposes and this can be run as part of manual cleaning.
- Each driver is responsible for providing the third party CI for testing the BIOS configuration.
- Tempest tests will be added using fake driver.

Upgrades and Backwards Compatibility

- Raise errors when there is no BIOSInterface support in driver.

Documentation Impact

- Documentation will be provided on how to configure a node for BIOS.
- API reference will be updated.
- Respective vendors should document the default BIOS values for reference.

References

5.15.137 HTTPBoot

https://bugs.launchpad.net/ironic/+bug/2032380

A long sought after feature in Ironic is to boot a node from a single HTTP URL. Except, there is more than one flavor of this functionality.

A first flavor is very virtual media like where we ask the Baseboard management controller to boot the machine from a URL. The exact details are vendor specific, but in essence UEFI firmware boots the OS from the URL.

The second flavor is very similar to PXE booting using DHCP, however the primary difference is through the use of a vendor class of HTTPClient is utilized instead of PXEClient. The funny thing, IPv6 network boot requires the same DHCP response in the form of a URL for the bootfile-url.

The reason to note both is they are both *very* similar, and both can be tested in our CI at this point. Previously we deferred implementation of the DHCP path because a lack of ability to test that in CI, but that is no longer a constraint of late 2023.

Implementing both would be relatively easy and very beneficial for the operator ecosystem, while also allowing for the navigation of the security and NAT incompatibility issues which exist with TFTP.

Problem description

Infrastructure operators need reliable, and relatively secure means of conveying bootloaders and initial artifacts to remote machines.

Ironics historical answer has been to utilize virtual media.

But not all hardware supports virtual media, and emulating a block device is not exactly the simplest thing once you boot an operating system. A helpful aspect with UEFI and the evolution of systems is now we have facilities in Baseboard management controllers and even UEFI base firmware support retrieving initial boot payload from a remote system using HTTP(s).

Proposed change

Given the similarity, it makes sense to implement support for both the BMC oriented path and the DHCP oriented path as part of single effort.

The greater benefit for implementing the DHCP oriented path is we can also extend this functionality to our downstream consumers with minimal effort.

BMC Path

- Update sushy-tools to support mapping cals to utilize a HTTP next boot URL to the virtual media driver code. Functionally this is similar, as there appears to be no means to inject the hint to boot from the URL into libvirt.
- Update sushy to support requesting to boot a node from a HTTP URL.
- Create a new `redfish-http-url` boot interface, named `RedfishHTTPBoot` BootInterface class based upon the underlying class `RedfishVirtualMediaBoot`. In this class, replace `_insert_vmedia`, and `_eject_vmedia` class. In essence these methods would perform the needful calls to the BMC to perform actions such as setting `BootSourceOverrideEnabled` to Once, `BootSourceOverrideMode` to UEFI, `BootSourceOverrideTarget` to UefiHttp, and finally `HttpBootUri` to the ISO file we wish to boot.

Constraints:

- Limited to UEFI.

Note

We may wish to retool some of the internals of the RedfishVirtualMediaBoot class for our implementation sanity.

Note

This interface with the BMCs is modeled around use of an ISO image as a boot source, where as the DHCP path is modeled upon a bootloader, much like iPXE.

DHCP Path

- Determine if we need to modify the `neutron-dhcp-agent`.
- Create an `httpboot` BootInterface based upon the existing `pxe` interface code base, and wire through a flag which is set by the PXE utils invocation of the DHCP code base, to signal the use of an HTTP(S) URL to the conductor. Specifically it would take the form of a flag on the pxe_utils method `prepare_instance_pxe_config` which would be supplied to the `dhcp_options_for_instance` method call in the same file. Likely as simple as `pxe_base` looking for a feature flag on the BootInterface class.

Note

We may wish to make an iPXE specific version of the boot interface as well, which is *also* already handled by a capabilities feature flag. A separation would enable Grub and iPXE use concurrently.

Alternatives

We could limit scope, but there really are not any alternatives, and both paths provide a great deal of functionality and benefit to users of Ironic.

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

openstack baremetal CLI

None, this is entirely a server side capability/configuration.

openstacksdk

None, this is entirely a server side capability/configuration.

RPC API impact

None

Driver API impact

No changes to the Driver API are anticipated, although this change functionally proposes two or three different BootInterfaces to be created.

Nova driver impact

None

Ramdisk impact

None. Booting the ramdisk would take the existing code paths with slight deviation where applicable for each driver case.

Security impact

The overall security posture of deployments could improve with these capabilities. Specifically UEFI firmware can boot an ISO or first stage bootloader over HTTPS.

Other end user impact

None

Scalability impact

None anticipated.

Performance Impact

None anticipated.

Other deployer impact

Deployers interested in using this functionality will have expanded operational and security capabilities which are in-line with established interfaces and data models in Ironic.

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
Julia (TheJulia) Kreger <juliaashleykreger@gmail.com>

Other contributors:
Volunteers welcome!

Work Items

- Add support to sushy and sushy-tools for the URL boot operation.
- Add support to pxe_utils logic to generate a URL boot response payload, and set that based upon a driver feature/capability flag.
- Compose lots of documentation.
- Create tempest suite to exercise both modes of boot operations.

Dependencies

IF we need to apply DHCP server configuration, similar to PXE/IPXE, chain loading attributes, then we will need to engage the Neutron developers.

Testing

The ideal path would be to create a single integration suite test in the ironic-tempest-plugin to set a node to utilize both interfaces, and toggle the nodes through a clean step, which would prove the interfaces work as we expect.

Upgrades and Backwards Compatibility

No issues are anticipated here.

Documentation Impact

We will likely need to compose more documentation than code in every case of these interface.

References

- https://www.dmtf.org/sites/default/files/standards/documents/DSP2053_2022.3.pdf
- https://bugs.launchpad.net/ironic/+bug/2032380
- https://en.opensuse.org/UEFI_HTTPBoot_Server_Setup

5.15.138 Huawei iBMC Driver

https://storyboard.openstack.org/#!/story/2004635

This specification proposes to add new interfaces that provide Ironic support to Huawei iBMC 2288H V5, CH121 V5 series servers.

Problem description

Huaweis Intelligent Baseboard Management System (iBMC) is an embedded server management system that is used to manage servers throughout their lifecycle. It provides a series of management tools for hardware status monitoring, deployment, energy savings, and security protection.

In addition to managing the nodes using IPMI protocol, this specification proposes to add hardware types and interfaces to manage Huawei servers using iBMC REST API.

Proposed change

New hardware type named *ibmc* will be added as part of this change. New power, management and vendor interfaces will be implemented for the *ibmc* hardware.

The interfaces use iBMC REST API to communicate with iBMC. The interfaces used are:

- iBMC.IBMCPower for Power operations
- iBMC.IBMCManagement for Management operations
- iBMC.IBMCVendor for Vendorspecific operations
- Power:

  This feature allows the user to turn the node on/off or reboot by using the power interface which will in turn call iBMC REST API.

- Management:

  This feature allows the user to get and set the primary boot device of the Huawei servers, and to get the supported boot devices.

- Vendor:

  This feature allows the user to perform vendor specific operations. For example, query the boot up sequence of the Huawei servers.

```
$ openstack baremetal node passthru call --http-method GET \
  <node id or node name> boot_up_seq
$ ["Pxe", "Hdd", "Cd", "Others"]
```

Alternatives

None

Data model impact

None

RPC API impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

Driver API impact

None

Nova driver impact

None

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Ramdisk impact

None

Other deployer impact

The following driver_info fields are required while enrolling node into Ironic:

- ibmc_address: The URL address to the ibmc controller, example: https://example.com
- ibmc_username: User account with admin/server-profile access privilege
- ibmc_password: User account password
- ibmc_verify_ca(optional): Whether to verify the host certificate or the path of a certificate file or directory with trusted certificates

Developer impact

None

Implementation

Assignee(s)

Primary assignee:

- QianBiao Ng (iampurse@vip.qq.com)
- Bill Chan (biaocy91@gmail.com)

Other contributors:
None

Work Items

- Add new iBMC hardware type, and adding new interfaces for Power, Management and Vendor.
- Writing appropriate unit tests to provide test coverage for iBMC driver.
- Writing configuration documents.
- Building a third party CI.

Dependencies

- Use python-ibmcclient library (not released) to communicate with HUAWEI iBMC REST API.

Testing

- Unit tests will be implemented for new iBMC driver.
- Third party CI will be provided.

Upgrades and Backwards Compatibility

None

Documentation Impact

- Updating Ironic documentation section *Enabling Drivers* with iBMC related instructions.

References

None

5.15.139 Define idrac Hardware Type Support of Redfish Interfaces

https://storyboard.openstack.org/#!/story/2004592

Operators need the ability to configure Ironic to use Redfish to manage Dell EMC bare metal servers, and be assured it:

- offers a means of incrementally increasing the use of Redfish as Ironics support, the standard, and Dell EMC service implementations evolve,
- delivers management protocol choice among those supported by Dell EMC Intelligent Platform Management Interface (IPMI), Redfish, and Web Services Management (WS-Man),
- provides all the `idrac` hardware type functionality they have relied on,
- works and will continue to,
- is supported by the vendor and community, and
- can offer Dell EMC value added.

This specification suggests the `idrac` hardware type provides that.

Problem description

Use cases

Expanding on what the introductory paragraph describes above, several use cases can be envisioned. While this specification enables them, it may turn out only some will find practical use among operators. That could be driven by many factors, including an existing versus greenfield deployment, operator comfort level with Redfish versus WS-Man, maturity of the protocol implementations, availability of needed functionality across Redfish and WS-Man, requirement for vendor value added, operational plans and schedules, among others.

Here they are. Note that except for the first two, they can be used in combination to configure a single Dell EMC bare metal server.

- An Admin User has a Dell EMC bare metal server and uses only WS-Man to manage it.

- An Admin User has a Dell EMC bare metal server and uses only Redfish to manage it.
- An Admin User has a Dell EMC bare metal server and uses both Redfish and WS-Man to manage it. When both offer the needed functionality, either is used.
- An Admin User has a Dell EMC bare metal server and uses both Redfish and WS-Man to manage it. Ironics ability to manage the server is maximized.
- An Admin User has a Dell EMC bare metal server and uses both Redfish and WS-Man to manage it. Vendor value added, which is available from only one or both, is used.

Proposed change

Background

The `idrac` hardware type is the Ironic driver intended for use with Dell EMC bare metal servers equipped with an integrated Dell Remote Access Controller (iDRAC) baseboard management controller (BMC). To date, all the out-of-band (OOB) management protocol-dependent interface implementations `idrac` has supported use the WS-Man protocol to interact with the iDRAC. Those implement the `inspect`, `management`, `power`, `raid`, and `vendor` hardware interfaces. Like the hardware type, they are named `idrac`. They rely on python-dracclients WS-Man client.

Operators also have the option to use the generic, vendor-independent `redfish` hardware type with Dell EMC bare metal servers that have an iDRAC that supports the Redfish protocol. `redfish`s supported OOB protocol- dependent interface implementations use the Redfish protocol to interact with the BMC. Those implement the `bios`, `inspect`, `management`, and `power` hardware interfaces. Again, like the hardware type, they are named `redfish`. They rely on sushys Redfish client. Importantly, while some of those work with the iDRAC, including `management` and `power`, not all of them do.

The `redfish` hardware type enables managing servers compliant with the Redfish protocol. However, it is relatively new, and the protocol standard has been evolving, along with its implementations by hardware vendors such as Dell EMC. As is common among standards, there is a difference between compliance and interoperability. For example, the Redfish `bios` interface implementation has not worked with the iDRAC because of client and server protocol implementation incompatibility.

While there is much functional overlap between the interface implementations supported by the `idrac` and `redfish` hardware types, it is not complete. Only `idrac` supports a `raid` interface implementation and only `redfish` supports `bios`. Also, the optional hardware interface functionality available in the `idrac` and `redfish` interface implementations can differ. For example, while the `redfish` implementation of the `management` hardware interface first introduced optional boot mode functionality, `idrac` does not offer that, yet. Therefore, those two hardware types are not perfect substitutes for one another.

Dell EMC wants to be able to offer its customers vendor value added as supported by the Redfish standard, like it has done through WS-Man. That benefits operators by making available features and functionality that has not yet been standardized. Dell EMC can be more responsive to its customers needs and differentiate itself in the market.

Goal

With this specification, we are going to achieve the goal of promoting and accelerating the adoption of Redfish by operators with Dell EMC bare metal servers.

Non-goals

The following is considered outside the scope of this specification:

- Support a node configuration with a mix of Redfish and WS-Man `management` and `power` interface implementations. The legacy `idrac` implementations of the `management` and `power` hardware interfaces interact to set the boot device. It is not clear there is a compelling need to accommodate that in a mixed Redfish and WS-Man configuration.
- The following TripleO command can be used to register and configure nodes for their deployment with Ironic:

  ```
  openstack overcloud node import instackenv.json
  ```

  See the TripleO register nodes documentation. It sets properties in a nodes `driver_info` field which are required by its driver. Presently, when the nodes driver is `idrac`, those are the properties `drac_address`, `drac_username`, and `drac_password` needed by the WS-Man interface implementations `idrac` has supported. See the iDRAC driver documentation.

  The Redfish interface implementations need similar, but different, properties in the `driver_info` field, including `redfish_address`, `redfish_system_id`, `redfish_username`, and `redfish_password`. See the Redfish driver documentation.

  Changing that TripleO command to set both the Redfish and WS-Man properties in a nodes `driver_info` field when its `driver` is `idrac` is beyond the scope of this specification. That will be addressed by a TripleO project blueprint.
- Define `idrac` hardware type support of IPMI interface implementations. That could be done as a follow-on to this.

Solution

This specification proposes to solve the problem it describes by changing the `idrac` hardware type. Since the Ironic Driver composition reform, we have been allowed to have one vendor driver with options configurable per node instead of many drivers for every vendor.[1] The reforms goals include[2]:

```
* Make vendors in charge of defining a set of supported interface
  implementations in priority order

* Allow vendors to guarantee that unsupported interface implementations will
  not be used with hardware types they define. This is done by having a
  hardware type list all interfaces it supports.
```

Implementing the solution in the `idrac` hardware type contributes toward making it the one Dell EMC driver for its bare metal servers with iDRACs and their value added implementations of the IPMI, Redfish, and WS-Man management protocols. It also aligns with the goals of the reform. That is what operators have come to expect.

Here are the details of the proposal.

- Define two new groups of interface implementations with entrypoints named `idrac-redfish` and `idrac-wsman`. The `idrac-redfish` entrypoints refer to Redfish interface implementations which are compatible with the iDRAC, presently those of the `management` and `power` hardware

[1] See the *introduction* paragraph of the Ironic Driver composition reform.
[2] See the *Introduction* subsection in the *Proposed change* section of the Ironic Driver composition reform.

interfaces. The `idrac-wsman` entrypoints are new names for the legacy `idrac` entrypoints. The legacy `idrac` entrypoints are left unchanged. For example:

```
ironic.hardware.interfaces.management =
    ...
    idrac = ironic.drivers.modules.drac.management:DracManagement
    idrac-redfish = ironic.drivers.modules.drac.
↪management:DracRedfishManagement
    idrac-wsman = ironic.drivers.modules.drac.
↪management:DracWSManManagement
    ...
    redfish = ironic.drivers.modules.redfish.management:RedfishManagement
```

- Declare `idrac` hardware type support for the `idrac`, `idrac-redfish`, and `idrac-wsman` interface implementations. `idrac` continues to have the highest priority by being first in its `supported_<INTERFACE>_interfaces` lists. Here `<INTERFACE>` is a type of hardware interface: `inspect`, `management`, `power`, etc. For example:

```
class IDRACHardware(generic.GenericHardware):
    ...
    @property
    def supported_management_interfaces(self):
        return [management.DracManagement, management.DracWSManManagement,
            management.DracRedfishManagement]
    ...
```

Note

The property uses classes, not instances nor entrypoint names. The example assumes the required modules are imported.

- New `idrac-redfish` entrypoints are defined by new Python classes, because using the generic, vendor-independent Redfish classes would make the `redfish` entrypoints synonyms for `idrac-redfish` and supported. A later requirement to change the name of an entrypoints Python class to resolve a Dell EMC-specific incompatibility or introduce vendor value added, which would eliminate support for `redfish`, could be a breaking change. The new Python classes are derived from the generic, vendor-independent Redfish classes.

- New `idrac-wsman` entrypoints are defined by new Python classes. Those classes are created by renaming the classes for the legacy `idrac` entrypoints from `Drac<INTERFACE>` to `DracWSMan<INTERFACE>`. Here `<INTERFACE>` refers to a type of hardware interface: `Inspect`, `Management`, `Power`, etc.

  The legacy `Drac<INTERFACE>` classes are redefined by simply deriving them from the new `DracWSMan<INTERFACE>` classes. For example:

```
class DracManagement(DracWSManManagement):
    pass
```

  That makes the legacy `Drac<INTERFACE>` classes aliases for the new `DracWSMan<INTERFACE>` classes. Any bug fixes or features added to the WS-Man interface implementations are available from both the `idrac` and `idrac-wsman` entrypoints. Having separate classes for the two groups of

entrypoints makes it possible to subsequently add logic that implements deprecation of the legacy `idrac` entrypoints by emitting a log message and similar.

Alternatives

- We could change the lowest layer of `python-dracclient` to support Redfish, in addition to WS-Man. However, we expect it would be challenging to provide `python-dracclient` APIs and workflows which abstract the very different Redfish and WS-Man technologies. Redfishs interface is RESTful, while WS-Man is a Simple Object Access Protocol (SOAP). APIs and workflows would likely need to be changed or newly defined. That would require substantial modification of the existing `idrac` interface implementations.
- We could maintain the status quo split of the `idrac` hardware type for WS-Man and `redfish` hardware type for Redfish. However, that would not promote and accelerate the use of Redfish among operators with Dell EMC bare metal servers today, because `redfish` does not offer everything `idrac` does. That also would not support resolving Dell EMC vendor- specific incompatibilities with the generic, vendor-independent `redfish` hardware type nor using Redfish to introduce vendor value added.
- We could let the `redfish` interface implementations use Redfish OEM extensions to address vendor-specific incompatibilities and introduce vendor value added. However, that seems inconsistent with the intent that they be generic and vendor-independent.

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

ironic CLI

None

openstack baremetal CLI

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

None

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

- A deployer can add `idrac-redfish` to the `enabled_management_interfaces` and `enabled_power_interfaces` options to enable those new interface implementations.
- A deployer can add `idrac-wsman` to the `enabled_inspect_interfaces`, `enabled_management_interfaces`, `enabled_power_interfaces`, `enabled_raid_interfaces`, and `enabled_vendor_interfaces` to enable those new interface implementations.
- A deployer must specify properties in the nodes `driver_info` field that are needed by Redfish interface implementations, including `redfish_address`, `redfish_system_id`, `redfish_username`, and `redfish_password`, to use the `idrac-redfish` interface implementations. That is in addition to the legacy properties the `idrac` hardware type has needed in `driver_info` `drac_address`, `drac_username`, and `drac_password`.

```
openstack baremetal node create --driver idrac --driver-info \
  drac_address=1.2.3.4 --driver-info drac_username=admin --driver-info \
  drac_password=password --driver_info redfish_address=https://1.2.3.4 \
  --driver-info redfish_system_id=/redfish/v1/Systems/System.Embedded.1 \
  --driver-info redfish_username=admin --driver-info \
  redfish_password=password
```

See the Redfish driver documentation, iDRAC driver documentation, and *Non-goals*.

- A deployer can specify the new `idrac-redfish` and `idrac-wsman` interface implementations on node enrollment:

```
openstack baremetal node create --driver idrac ... --management-interface␣
↪\
  idrac-wsman --power-interface idrac-wsman ...
```

They can also be set by the following command:

```
openstack baremetal node set <NODE> --management-interface idrac-redfish \
  --power-interface idrac-redfish
```

They must be enabled as described above.

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
rpioso

Other contributors:
None

Work Items

- Define two new groups of interface implementations with entrypoints named `idrac-redfish` and `idrac-wsman`.
- Declare `idrac` hardware type support for the `idrac`, `idrac-redfish`, and `idrac-wsman` interface implementations.
- Integration test the changes against Dell EMC bare metal servers.
- Modify the Dell EMC Ironic third-party continuous integration (CI) to cover supported configurations added by this specification.
- Update the iDRAC driver documentation.

Dependencies

This specification is related to the Driver composition reform.

It specifically targets Dell EMC bare metal servers equipped with an iDRAC and managed by the `idrac` hardware type.

Testing

This is not testable in the gate given current limitations on the availability of the specific hardware required.

The mitigation plan is to add coverage to the Dell EMC Ironic third-party CI for supported configurations added by this specification that we expect to be common.

Upgrades and Backwards Compatibility

This change is designed to be backwards compatible. The legacy `idrac` interface implementation entrypoints will be supported for at least some time. A separate story will cover their deprecation.

We will recommend switching to the appropriate new `idrac-redfish` and `idrac-wsman` interface implementation entrypoints as soon as it is possible.

Documentation Impact

The iDRAC driver documentation is updated to:

- describe switching from the legacy `idrac` interface implementation entrypoints to the new `idrac-redfish` and `idrac-wsman` entrypoints,
- reflect the changes to the supported interface implementations, and
- inform that a node configuration with a mix of Redfish and WS-Man `management` and `power` interface implementations is not supported.

References

OpenStack software projects:

- ironic
- python-dracclient
- sushy

Related Ironic specifications:

- Driver composition reform

Documentation:

- iDRAC driver
- Redfish driver
- TripleO register nodes

Standards:

- IPMI
- Redfish
- WS-Man

5.15.140 Out-of-band disk-erase for Gen10 and above HPE Proliant Servers

https://storyboard.openstack.org/#!/story/2004786

This specification proposes implementation of out-of-band disk-erase for iLO5 managed HPE Proliant servers.

Problem description

In the current scenario where disk-erase on HPE Proliant servers is done only via inband cleaning, iLO5 based HPE Proliant Gen10 servers provide support to perform out-of-band disk-erase which was not there in Gen9 and older servers. However, disk-erase request would be accepted by iLO only when system boot completes POST. Hence disk-erase needs to be accompanied by a reboot.

Proposed change

This spec proposes to implement out-of-band disk-erase clean_step in hardware type `ilo5` under new management interface `Ilo5Management` which would be inherited from existing management interface `IloManagement`.

List of changes required:

- The following would be the composition of the new management interface `Ilo5Management`:
  - `erase_devices` - This will erase all disks on the baremetal node.
    - *erase_devices* will call proliantutils library method *do_disk_erase* to perform the operation in iLO. User can also choose between different erase pattern (ex. block, overwrite, crypto, zero) to perform the disk erase operation.
    - The reboot is required to initiate the disk erase. The actual disk erase operation would take time based on disk type and size.

Alternatives

One can perform in-band disk-erase to achieve the same result. However, The ramdisk to be used in such case should have proliant-tools element that bundles ssacli utility required for disk-erase operations as part of the image.

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

None

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

Users need to configure two options to make use of OOB disk-erase on HPE Proliant Gen10 servers.

- Configure the hardware type `ilo5` to ([DEFAULT] `enabled_hardware_types`).
- Configure the new management interface `ilo5` to ([DEFAULT] `enabled_management_interfaces`).

Developer impact

None

Implementation

Assignee(s)

Primary assignee: pareshsao

Work Items

- Add a new management interface `Ilo5Management` to hardware type `ilo5`
- Writing unit-test cases for the new OOB disk-erase interface.

Dependencies

Support for OOB disk-erase in proliantutils is under development and is yet to be released.

Testing

Unit test cases will be added. Will be tested in 3rd party CI setup.

Upgrades and Backwards Compatibility

None

Documentation Impact

Need to update iLO driver documentation for new management interface.

References

None

5.15.141 Out-of-band RAID configuration for Gen10 and above HPE Proliant Servers

https://bugs.launchpad.net/ironic/+bug/1716329

This specification proposes implementation of out-of-band RAID configuration for ILO managed HPE Proliant servers.

Problem description

In the current scenario where RAID configuration on HPE Proliant servers is done only via inband cleaning, Ilo5 based HPE Proliant Gen10 servers provide support to perform out-of-band RAID configuration which was not there in Gen9 and below servers. However, the raid creation or deletion will take into effect only when the system reaches POST stage. Hence, creation or deletion of RAID needs to be accompanied by a reboot.

Proposed change

This spec proposes to implement out-of-band RAID configuration as described by the parent spec [1]. This will require the implementation of a new hardware type `Ilo5Hardware` and a new raid interface for ilo as `IloRAID`.

OOB RAID configuration will be a four step process. 1. delete_configuration - delete the current raid config from the system. 2. read_configuration - get the updated raid config from system and update the node properties accordingly. 3. create_configuration - create the raid config which set by the user in target_raid_config of node properties. 4. read_configuration - get the updated raid config from system and update the node properties accordingly.

List of changes required:

- The following would be the composition of `Ilo5Hardware`:
  - This hardware type would be supported on ilo5 based HPE Proliant servers.
  - Ilo5Hardware will inherit all interfaces of parent class IloHardware.
  - Ilo5Hardware will support the new RAID interface `IloRAID`.

- The following would be the composition of `IloRAID`:
  - IloRAID will inherit RAIDInterface of base class.
  - `delete_configuration` - This will delete the RAID configuration on the bare metal node.
    - Since a reboot is required for changes to get reflected, this function will be decorated with additional argument *reboot_required* with value set to *True*.
    - It will create an IloClient object from proliantutils library to do operations on the iLO. This will make call to delete_raid_configuration of proliantutils library to delete the logical drives on the system.
  - `create_configuration` - This will create the RAID configuration on the bare metal node.
    - Since a reboot is required for changes to get reflected, this function will be decorated with additional argument *reboot_required* with value set to *True*.
    - It will create an IloClient object from proliantutils library to do operations on the iLO. This will make call to create_raid_configuration of proliantutils library to place a request to firmware to create the logical drives on the system.
  - `read_configuration` - This will read the RAID configuration on the bare metal node.
    - It will create an IloClient object from proliantutils library to do operations on the iLO. This will make call to read_raid_configuration of proliantutils library to get the logical drives on the system. Hence, it will update the node properties with the actual RAID configuration when called after `create_configuration` and to `None` when called after `delete_configuration`.
- The following would be the updates required in cleaning architecture in ironic to support post reboot operation if required any clean step.
  - Addition a new boolean positional argument `reboot_required` to clean_step function of BaseInterface. Default is set to `False` for this parameter. NOTE: The same approach is being used in inband cleaning for steps that require reboot.
  - Update ironic/conductor/manager.py:_do_next_clean_step() for each step to call prepare_cleaning() if reboot_required is set to True and result of the last command interface.execute_clean-step() is not `clean wait`.

Alternatives

One can perform in-band raid configuration to achieve the same result. However, The ramdisk to be used in such case should have proliant-tools element that bundles ssacli utility required for RAID operations as part of the image.

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

None

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

User need to configure below two things to make use of OOB RAID configuration on HPE Proliant Gen10 servers.

- Configure the new hardware type `ilo5` to ([DEFAULT] `enabled_hardware_types`).
- Configure the new raid interface `ilo5` to ([DEFAULT] `enabled_raid_interfaces`).

Developer impact

None

Implementation

Assignee(s)

Primary assignee: theanshuljain

Work Items

- Add a new hardware type for ilo `Ilo5Hardware` which inherits IloHardware.
- Add a new hardware interface `IloRAID` which inherits base.RAIDInterface.
- Writing unit-test cases for the new OOB RAID interface.

Dependencies

Support for OOB RAID in proliantutils is under development and is yet to be released.

Testing

Unit test cases will be added. Will be tested in 3rd party CI setup.

Upgrades and Backwards Compatibility

None

Documentation Impact

Need to update iLO driver documentation for new hardware type and RAID interface.

References

[1] Ironic generic raid spec: https://review.opendev.org/173214

5.15.142 UEFI iSCSI Boot for iLO drivers

https://bugs.launchpad.net/ironic/+bug/1526861

HPE ProLiant Servers (Gen9 and beyond) supports UEFI iSCSI Boot through its firmware. The proposed feature is to add support for this firmware based booting of an iSCSI Cinder volume in UEFI mode for Ironic iLO drivers.

Problem description

Currently, Ironic has ability to boot from Cinder volume. Moreover, this support is to boot from an iSCSI volume using bootloaders like iPXE. It doesnt provide any way to harness the feature of some servers which inherently supports booting from an iSCSI volume using their firmware capabilities. Hardware can be configured programmatically to boot from an iSCSI volume through firmware.

Proposed change

This change is based on the reference driver implementation guidelines proposed by Boot from Volume - Reference Drivers spec to support booting ironic nodes from a storage device that is hosted and/or controlled remotely. This change proposes two new methods for iLO drivers management interface; namely `set_iscsi_boot_target` and `clear_iscsi_boot_target`, which will facilitate setting and clearing iSCSI target information using iLO interfaces for UEFI iSCSI boot capable HPE Proliant servers.

The boot interface method `prepare_instance()` in `ilo` hardware type will check if the instance requested boot mode is UEFI and given volume is bootable. If so, it will set the iSCSI target in the iLO and set boot device to iSCSI target.

If the instance requested boot mode is BIOS the behavior for the two boot interfaces (`ilo-pxe` and `ilo-virtual-media`) will be as under:

- `ilo-pxe` : It will fallback to iPXE to boot the volume.
- **`ilo-virtual-media`: It will throw the following error:**
  `virtual media cannot boot volume in bios.`

The function definition for `ilo-pxe` boot interface with its pseudo-code will be as follows:

```
class IloPXEBoot(pxe.PXEBoot):

  def prepare_instance(self, task):
    """Prepares the boot of instance.

    :param task: a task from TaskManager.
    :returns: None
    :raises: IloOperationError, if some operation on iLO failed.
    """
    if deploy_utils.is_iscsi_boot(task) and boot_mode == 'uefi':
      #Call the management interface
      task.driver.management.set_iscsi_boot_target(task)
      #Set boot device to 'ISCSIBOOT'
      deploy_utils.try_set_boot_device(task, boot_devices.ISCSIBOOT)

    else:
      #Let iPXE handle this
      super(IloPXEBoot, self).prepare_instance(task)

  def clean_up_instance(self, task):
    """Cleans up the boot of instance.

    :param task: a task from TaskManager.
    :returns: None
    :raises: IloOperationError, if some operation on iLO failed.
    """
    if deploy_utils.is_iscsi_boot(task) and boot_mode == 'uefi':
      #Call the management interface
      task.driver.management.clear_iscsi_boot_target(task)

    else:
      #Let iPXE handle this
```

(continues on next page)

(continued from previous page)

```
        super(IloPXEBoot, self).clean_up_instance(task)
```

The function definition for `ilo-virtual-media` boot interface with its pseudo-code will be as follows:

```
class IloVirtualMediaBoot(base.BootInterface):

  def prepare_instance(self, task):
    """Prepares the boot of instance.

    :param task: a task from TaskManager.
    :returns: None
    :raises: IloOperationError, if some operation on iLO failed.
    """
    if deploy_utils.is_iscsi_boot(task) and boot_mode == 'uefi':
      #Call the management interface
      task.driver.management.set_iscsi_boot_target(task)
      #Set boot device to 'ISCSIBOOT'
      deploy_utils.try_set_boot_device(task, boot_devices.ISCSIBOOT)
      return

    elif deploy_utils.is_iscsi_boot(task) and boot_mode == 'bios':
      #Throw the error in bios boot mode
      msg = 'virtual media can not boot volume in bios mode.'
      raise exception.InstanceDeployFailure(msg)

    else:
      #Default code

  def clean_up_instance(self, task):
    """Cleans up the boot of instance.

    :param task: a task from TaskManager.
    :returns: None
    :raises: IloOperationError, if some operation on iLO failed.
    """
    if deploy_utils.is_iscsi_boot(task) and boot_mode == 'uefi':
      #Call the management interface
      task.driver.management.clear_iscsi_boot_target(task)
    else:
      #Fall to virtual media cleanup
```

Two new methods will be added in `ilo` drivers management interface `ilo.management.IloManangement`: * set_iscsi_boot_target() - To set iSCSI target information into iLO * clear_iscsi_boot_target() - To clear iSCSI target information from iLO

New version of proliantutils library would be released that supports the above mentioned methods.

The function definition with its pseudo-code will be as follows:

```
class IloManagement(base.ManagementInterface):

  def set_iscsi_boot_target(self, task):
    """Set iscsi boot volume target info from the node.

    :param task: a task from TaskManager.
    """
    #Proliants call to set iscsi target info

  def clear_iscsi_boot_target(self, task):
    """Clear iscsi boot volume target info from the node.

    :param task: a task from TaskManager.
    """
    #Library call to clear iscsi target info
```

Alternatives

None.

Data model impact

None.

State Machine Impact

None.

REST API impact

None.

Client (CLI) impact

None.

RPC API impact

None.

Driver API impact

None.

Nova driver impact

None.

Security impact

None.

Ramdisk impact

None.

Other end user impact

None.

Scalability impact

None.

Performance Impact

None.

Other deployer impact

Deployers will be able to configure server which support UEFI iSCSI boot with this change. The workflow will be as follows:

- Operator configures the node with appropriate hardware type with boot interface and adds the capability `iscsi_boot=true` in `node.properties['capabilities']` (or it could be populated by inspection, but its not part of this spec on how it gets populated).
- Operator creates a flavor with Compute capability `iscsi_boot=true` to request bare metal booting from Cinder volume.
- Tenant creates a Cinder volume.
- Tenant requests a bare metal instance to be booted up with a Cinder volume with the above mentioned flavor.
- Node having ilo-virtual-media as boot interface with capability iscsi_boot=true should also have capability boot_mode configured to uefi only.
- Nova Ironic virt driver passes information about iSCSI volume to Ironic. For more information, refer ironic spec Add volume connection information for Ironic nodes.

Developer impact

None.

Implementation

Assignee(s)

Primary assignee:
kesper

Other contributors:
deray stendulker

Work Items

- Need to add changes in `ilo-pxe` and `ilo-virtual-media` boot interfaces.
- Need to implement `set_iscsi_boot_target` and `clear_iscsi_boot_target` in `ilo` management interface.

Dependencies

None.

Testing

This feature would be tested using HPE iLO third-party CI.

Upgrades and Backwards Compatibility

None.

Documentation Impact

iLO drivers documentation will be updated for this feature.

References

- Boot from Volume - Reference Drivers

5.15.143 Implement Rescue Mode

https://bugs.launchpad.net/ironic/+bug/1526449

Implement Nova rescue/unrescue in Ironic. Also implement an extension in IPA that carries out rescue-related tasks. After rescuing a node, it will be left running a rescue ramdisk, configured with the rescue_password, and listening with ssh on the specified network interfaces.

Problem description

Ironic does not currently implement the Nova rescue/unrescue interface. Therefore, end users are left with few options for troubleshooting or fixing anomalous and misconfigured nodes.

Proposed change

- Implement rescue(), and unrescue() in the Ironic virt driver (no spec reqd): https://blueprints.launchpad.net/nova/+spec/ironic-rescue-mode
- Add InstanceRescueFailure, and InstanceUnRescueFailure exceptions to Nova
- Add methods to Nova driver to poll Ironic to wait for nodes to rescue and unrescue, as appropriate.
- Store plaintext password in Ironic node instance_info for use on the rescued node.
- Add method for injecting password into OS.
- Modify Ironic state machine as described in the state machine impact section
- Add AgentRescue driver (implements base.RescueInterface). This implementation will be available only through hardware type. It would not be supported in the classic drivers as they would be deprecated shortly.

- For classic drivers rescue interface would be set to None. An exception of type UnsupportedDriverExtension would be generated when any methods of this interface are invoked for nodes managed by classic drivers.
- Add periodic task _check_rescue_timeouts to fail the rescue process if it takes longer than rescue_callback_timeout seconds for the rescue ramdisk to come online.
- Add Conductor methods: do_node_rescue, and do_node_unrescue
- Add Conductor RPC calls: do_node_rescue, and do_node_unrescue (and increment API version)
- Add conductor.rescue_callback_timeout config option
- Add rescue-related functionality to Ironic Python Agent including ability to set rescue password and kick off any needed network configuration automation
- Documentation of good practices for building rescue ramdisk in multitenant environments
- Add rescue_network configuration, which contains the UUID of the network the rescue agent should be booted onto. For security reasons, this should be separate from the networks used for provisioning and cleaning in multi-tenant environments.

An outline of the standard (non-error) rescue and unrescue processes follows:

Standard rescue process:

1. User calls Nova rescue() on a node.
2. Nova ComputeManager calls the virt drivers rescue() method, passing in rescue_password as a parameter.
3. Virt driver calls node.set_provision_state(RESCUE), with the rescue_password as a parameter.
4. Virt driver loops while waiting for provision_state to change, and updates Nova state as appropriate.
5. Ironic API receives set_provision_state call, and performs do_node_rescue RPC call (ACTIVE -> RESCUING).
6. Ironic conductor sets rescue password in instance_info and hands off call to appropriate driver.
7. Driver boots rescue ramdisk (RESCUING -> RESCUEWAIT), using the configured boot driver. As part of this process, Ironic will put the node onto the rescue_network, as configured in ironic.conf.
8. Agent ramdisk boots, performs a lookup (/v1/lookup in ironic-api), gets node info back, and begins heartbeating (/v1/heartbeat in ironic-api).
9. Upon receiving heartbeat, the conductor calls finalize_rescue (/v1/commands) with config drive and rescue password (RESCUEWAIT -> RESCUING), and removes the rescue password from the instance_info, as its no longer needed.
10. Agent sets password, configures network from information in config drive, and stops agent service.
11. The conductor flips network ports putting the node back on the tenant network, and the state is set to RESCUE.

Standard Unrescue process:

1. User calls Nova unrescue() on a node.
2. Nova calls Ironic unrescue() virt driver.
3. Virt driver calls node.set_provision_state(ACTIVE).

4. Virt driver loops while waiting for provision_state to change, and updates Nova state as appropriate.
5. Ironic API receives set_provision_state call, and performs do_node_unrescue RPC call.
6. Ironic conductor hands off call to appropriate driver.
7. Driver performs actions required to boot node normally, and sets provision state to ACTIVE.

Rescue/Unrescue with standalone Ironic:

1. Call Ironic provision state API with verb rescue, with the rescue password as an argument.
2. When finished with rescuing the instance, call Ironic provision state API with unrescue verb

Alternatives

- Continue to not support rescue and unrescue.
- Use console access to get rescue-like access into the OS, although this may not help in cases of lost password.

Data model impact

Essentially none. We will use instance_info to store, and subsequently retrieve, the rescue_password while rescuing a node.

State Machine Impact

- Add states to the Ironic state machine: RESCUING, RESCUEWAIT, RESCUE, RESCUEFAIL, UNRESCUING, UNRESCUEFAIL.
- Add transitions to the Ironic state machine:
  - ACTIVE -> RESCUING (initiate rescue)
  - RESCUING -> RESCUE (rescue succeeds)
  - RESCUING -> RESCUEWAIT (optionally, wait on external callback)
  - RESCUING -> RESCUEFAIL (rescue fails)
  - RESCUEWAIT -> RESCUING (callback succeeds)
  - RESCUEWAIT -> RESCUEFAIL (callback fails or abort issued)
  - RESCUEWAIT -> DELETING (delete instance without waiting)
  - RESCUE -> RESCUING (re-rescue node)
  - RESCUE -> DELETING (delete rescued instance)
  - RESCUE -> UNRESCUING (unrescue node)
  - UNRESCUING -> UNRESCUEFAIL (unrescue fails)
  - UNRESCUING -> ACTIVE (unrescue succeeds)
  - UNRESCUEFAIL -> RESCUING (re-rescue node after failed unrescue)
  - UNRESCUEFAIL -> UNRESCUING (re-unrescue node after failed unrescue)
  - UNRESCUEFAIL -> DELETING (delete instance that failed unrescuing)

  - RESCUEFAIL -> RESCUING (re-rescue after rescue failed)
  - RESCUEFAIL -> UNRESCUING (unrescue after failed rescue)
  - RESCUEFAIL -> DELETING (delete instance after failed rescue)
- Add state machine verbs:
  - RESCUE
  - UNRESCUE

REST API impact

Modify provision state API to support the states and transitions described in this spec. Also increment the API microversion. Nodes in states introduced by this spec (and related, future microversion) would be unable to be modified by clients using an earlier microversion.

Client (CLI) impact

ironic CLI

None, as this CLI is anticipated to be deprecated prior to this feature landing.

openstack baremetal CLI

The OSC command line will require additional code to handle the new state. New command line options `rescue` and `unrescue` would be added to support rescue and unrescue operations.

RPC API impact

Add do_node_rescue and do_node_unrescue to the Conductor RPC API.

Driver API impact

Add a new method clean_up() for RescueInterface in base.py. This method would perform any necessary clean up of the node upon RESCUEWAIT timeout/failure or finishing rescue operation. Some of the cleaning tasks are removing rescue password from the node. Ramdisk boot environment should be cleaned if ironic is managing the ramdisk boot. It would have default implementation as given below:

```
class RescueInterface(BaseInterface):
    def clean_up(self, task):
        pass
```

Nova driver impact

Implement rescue() and unrescue() in the Nova driver. Add supporting methods including _wait_for_rescue() and _wait_for_unrescue().

Ramdisk impact

An agent that wishes to support rescue should:

- Read and understand ipa-api-url kernel parameter for configuring API endpoint

- **Implement a client for ironics lookup API call**
  - The rescue_password will be in instance_info in the node object returned by Ironic on lookup. This can be placed in a linux-style /etc/shadow entry to enable a new user account.
- **Implement heartbeating to the appropriate API endpoint in Ironic**
  - After one heartbeat, the agent should then kickoff any action needed to reconfigure networking, such as re-DHCPing, as the Ironic conductor will complete all actions to finish rescue - including moving the node off a network with access to Ironic API, if relevant.
  - Once network is reconfigured, the agent process should shutdown. Rescue is complete.

IPA will have a rescue extension added, implementing the above functionality.

Security impact

The rescue_password must be sent from Nova to Ironic, and thereafter to the rescued node. If, at any step in this process, this password is intercepted or changed, an attacker can gain root access to the rescued node.

Additionally, the lookup endpoint will be required to return the rescue password as a response to the first lookup once rescue is initiated. That means a properly executed timing attack could recover the password, but since this would also cause the rescue to fail (despite the node changing states), its at worst a denial of service.

Security vulnerabilities involving the rescue ramdisk is another source of attacks. This is different from existing ramdisk issues, as once the rescue is complete, the tenant would have access to the ramdisk. This means deployers may need to ensure no secret information (such as custom cleaning steps or firmwares) are not present in the rescue ramdisk.

IPA is entirely unauthenticated. If IPA endpoints continue to be available after a node is rescued, then attackers with access to the tenant network would be able to leverage IPAs REST API to gain privileged access to the host. As such, IPA itself should be shut down, or the network should be sufficiently isolated during rescue operations.

Other end user impact

We will add rescue and unrescue commands to OSC Client.

Scalability impact

None.

Performance Impact

None.

Other deployer impact

Add conductor.rescue_callback_timeout config option.

Multi-tenant deployers will most likely need to support two ram disksone running IPA for use with normal node-provisioning tasks, and another running IPA for rescue mode (with non-rescue endpoints disabled). This is to ensure the full suite of tooling and authentication needed for secure cleaning is not given to a tenant.

Additionally, in some environments, operators may not want to use the full Ironic Python Agent inside the rescue ramdisk, due to its requirement for python or linux-centric nature. They may use statically compiled software such as onmetal-rescue-agent[0] to perform the lookup and heartbeat needed to finalize cleaning.

Developer impact

None.

Implementation

Assignee(s)

Primary assignee:
JayF

Other contributors:
Shivanand Tendulker (stendulker) Aparna (aparnavtce)

Work Items

See proposed changes.

Dependencies

- Updating the Ironic virt driver in Nova to support this.

Testing

Unit tests and Tempest tests must be added.

Upgrades and Backwards Compatibility

Clients that are unaware of rescue-related states may not function correctly with nodes that are in these states.

Documentation Impact

Write documentation.

References

5.15.144 In-band Deploy Steps

https://storyboard.openstack.org/#!/story/2006963

[0] https://github.com/rackerlabs/onmetal-rescue-agent

Since the Rocky release, Ironic has had support for deploy steps. These allow drivers to customise the node deployment flow. Currently these steps are limited to out of band execution - steps may not be executed on the node via IPA during deployment. This spec proposes to support execution of in-band deploy steps, in a similar manner to in-band cleaning steps.

Example use cases:

- Software RAID configuration during deployment
- In-band BIOS configuration during deployment

Problem description

Ironics deployment process has historically been quite rigid, not allowing for much customisation outside of the scope of the supported features. This is changing, starting with deploy steps, drivers are able to define custom tasks to perform during deployment using the pattern established for cleaning. The deploy templates feature exposes these deploy steps to users of nova via traits applied to flavors or images.

There are some limitations to deploy steps that we aim to address in this spec:

- deploy steps cannot be executed in-band (on the node, via IPA) during deployment
- deploy steps may only be executed before or after what we will refer to here as the mega step

As we will see, these two issues are linked, since we must break apart the mega step in order to support in-band deploy steps.

Mega step

The mega step starts with the node powered off and configured to boot on the provisioning network. When it has finished, the image has been written to disk, the boot device configured, and the node is plugged into the tenant network and powered on. This covers a significant portion of the deployment process.

Use cases

The following are some use cases for in-band deploy steps.

- as a user of ironic, I want a machine configured with a particular software RAID layout during deployment
- as a user of ironic, I want a machine configured with particular BIOS settings during deployment
- as a user of ironic, I want custom firmware installed during deployment
- as a user of ironic, I want to apply custom NIC configuration during deployment

And here are some for decomposing the mega step.

- as an ironic driver maintainer, I want to create a deploy step that executes with the node booted on the provisioning network
- as an ironic driver maintainer, I want to create a deploy step that executes after the image has been written to disk, while the node is attached to the provisioning network and powered on

Proposed change

Mega step context

The mega step is actually a deploy step called `deploy` on the `deploy` interface. In order to understand how it will be changed, we must first understand what it does and how it fits into the overall deployment flow. Some details in the following will depend on the particular drivers and configuration in use, but we will try to cover the most common usage.

We start our story in the ironic conductor, where a `do_node_deploy` RPC has been received.

1. `do_node_deploy` RPC received:
    1. validation of `power` and `deploy` interfaces, traits, and deploy templates
    2. the nodes provision state is set to `deploying` (or `rebuilding` if `rebuild` is `True`)
2. execution continues in a new worker thread:
    1. config drive is built and stored, if using one
    2. the `prepare` method of the `deploy` interface is called
    3. deploy steps for execution are determined based on drivers and deploy templates, then stored in `driver_internal_info.deploy_steps`
    4. trigger execution of the next deploy step
3. for each deploy step:
    1. update `driver_internal_info.deploy_step_index` to the index of the current step
    2. execute the deploy step
        1. if it returns `DEPLOYWAIT`, this is an asynchronous step. Exit the worker thread and wait for a `continue_node_deploy` RPC
        2. if it returns `None`, continue to execute the next deploy step
4. if all deploy steps complete:
    1. clean up `driver_internal_info`
    2. start the nodes console, if necessary
    3. the nodes provision state is set to `active`

Well that doesnt look too bad until we realise theres some complexity and asynchronism hidden in there.

Prepare

First of all, lets look at the `prepare` method of the `deploy` interface. For the `agent` and `iscsi` deploy interfaces, this involves:

1. power off the node
2. remove tenant networks (for rebuild, when writing an image)
3. add provisioning network (when writing an image)
4. attach volumes
5. prepare ramdisk boot

So we know that if an image is to be written to the node, then after `prepare`, the node will be ready to boot into the IPA ramdisk on the provisioning network.

Continue node deploy RPC

The next item to unpack here is asynchronous steps and the `continue_node_deploy` RPC. While waiting for execution of an asynchronous step, the nodes provision state is set to `wait callback`. Completion of the step is checked for either in the driver via a periodic task (the driver sets `driver_internal_info.deployment_polling`), or in the heartbeat handler for IPA. On completion, the `continue_node_deploy` RPC is triggered, and the node returns to the `deploying` provision state.

Mega step

The final piece here is the mega step itself: the `deploy` method on the `deploy` interface.

1. reboot the node, wait for heartbeat (when writing an image)
2. on the first IPA heartbeat:
    1. write image to disk synchronously (`iscsi` deploy interface)
    2. call IPA `standby.prepare_image` API, wait for heartbeats until complete (`direct` deploy interface)
3. prepare instance to boot
4. power off the node
5. remove provisioning network
6. configure tenant networks
7. power on the node

At this point, any deploy steps with a lower priority than the mega steps (100) will be executed. Care is required at this point however, since the node is already attached to the tenant network and booting.

In-band cleaning steps

Here is a quick overview of how in-band cleaning works for reference.

In-band cleaning starts by configuring the node on the provisioning network and booting up the IPA ramdisk. On the first heartbeat, the list of in-band steps is queried, combined with out-of-band steps and stored in `driver_internal_info.clean_steps`.

In-band clean steps are advertised by the `deploy` interface, which overrides the `execute_clean_step` method to execute them via the IPA API. In-band clean steps are always asynchronous, with polling triggered via the IPA heartbeat.

In-band clean steps may request that the node is rebooted after completion of the step via a `reboot_requested` flag in their step definition. There is some handling of the case where an IPA returns with a different version after reboot. In this case automated cleaning is restarted, and manual cleaning is aborted.

One final detail is the ability to define hooks that execute after completion of an in-band cleaning step via the `@post_clean_step_hook` decorator.

Observations

In order to gather a list of in-band deploy steps, we need to be able to communicate with IPA. This is first possible at step 2 of the mega step.

We may wish to still allow execution of out-of-band deploy steps before IPA has booted, to avoid unnecessary delays. An example here is BIOS or RAID configuration on Dell iDRACs, which can require the node to be powered on for the lifecycle controller to perform the configuration jobs. An additional boot cycle adds significant delay to the deployment process. This would represent a divergence in behaviour between deployment and cleaning.

When booting from a volume, there may be no image to write. Currently in that case, IPA is not used. This would prevent the use of in-band deploy steps, but booting up IPA just to gather a list of deploy steps would increase the time required to boot from a volume.

The above description of deployment did not cover fast track deploys. This feature allows a node that is already booted with an IPA ramdisk, e.g. from discovery, to bypass the reboot in the mega step.

Proposed mega step decomposition

The following describes the proposed decomposition of the mega step into separate steps.

1. `deploy` [100]:
    1. reboot the node, wait for heartbeat (when writing an image?)
    2. gather in-band deploy steps from the agent
2. `write_image` [80]:
    1. write image to disk synchronously (`iscsi` deploy interface)
    2. in-band deploy step that does the equivalent of the `standby.prepare_image` IPA API and waits for completion of the write (`direct` deploy interface)
3. `prepare_instance_boot` [60]:
    1. install bootloader (if needed)
    2. configure the boot interface
4. `tear_down_agent` [40]:
    1. power off the node
5. `switch_to_tenant_network` [30]:
    1. remove provisioning network
    2. add tenant networks
6. `boot_instance` [20]:
    1. power on the node

The useful priority ranges for inserting custom in-band steps are:

- 99 to 81: preparation before writing the image (e.g. software RAID).
- 79 to 61: modifications to the image before a bootloader is written (e.g. GRUB defaults changes).
- 59 to 41: modifications to the final instance (e.g. software configuration).

deploy

Priority: 100

This deploy step will be largely unchanged from the current `deploy` step. Changes will be necessary for fast track deploys, to skip the direct call to `continue_deploy` and rely on the new deploy steps. For boot from volume this step currently performs the tenant network configuration, instance preparation and reboot, however that can also be moved to the new steps.

write_image

Priority: 80

For the `iscsi` interface, the `continue_deploy` method will be split into a `write_image` deploy step and the `prepare_instance_boot`, `tear_down_agent`, `switch_to_tenant_network`, and `boot_instance` deploy steps.

For the `direct` interface, this step will start as an out-of-band one, will collect the necessary information, then switch into being executed in-band by IPA. It will be equivalent to executing the existing `standby.prepare_image` command via the agent API, and will block until the image has been written. This allows us to remove this special case of command status polling. There will need to be a transition period to support old IPA ramdisks that do not support in-band deploy steps.

prepare_instance_boot

Priority: 60

This will be largely equivalent to the `prepare_instance_to_boot` method of the `AgentDeployMixin`.

tear_down_agent

Priority: 40

In this step, the node will be powered off and ramdisk boot environment will be removed.

switch_to_tenant_network

Priority: 30

In this step, the node will be switched from the provisioning network to tenant networks.

boot_instance

Priority: 20

In this step, the node will be powered on.

Agent heartbeat handler

The `heartbeat` method of the `HeartbeatMixin` currently provides an extension of the logic of the `deploy` step. This includes calling `continue_deploy` on the first heartbeat, and `reboot_to_instance` when the deployment is finished. This logic will be unnecessary with these methods as deploy steps, but will remain in place for a period for backwards compatibility. Drivers will advertise support for decomposed deploy steps by returning `True` from a method called `has_decomposed_deploy_steps`.

Proposed in-band deploy step support

In-band deploy steps will be handled in a similar way to in-band cleaning steps, with some differences:

- out-of-band deploy steps with a priority greater than 100 may be executed before the node has booted up
- the `deploy` interface may provide both in-band and out-of-band deploy steps
- there is no equivalent of manual cleaning
- IPA version mismatch will lead to termination of deployment

In-band deploy steps must have a priority between 41 and 99 to ensure they execute after `deploy` and before `tear_down_agent`.

In-band deploy steps will be driven through the agents heartbeat mechanism. The first heartbeat will query the in-band steps, combine them with out-of-band steps and store them in `driver_internal_info.deploy_steps`.

In-band deploy steps are advertised by the `deploy` interface, which will override the `get_deploy_steps` method to query the steps from IPA, and the `execute_deploy_step` method to execute them via the IPA API. This will be slightly different from clean steps, to support execution of out-of-band steps on the `deploy` interface. In-band deploy steps are always asynchronous, with polling triggered via the IPA heartbeat.

In-band deploy steps may request that the node is rebooted after completion of the step via a `reboot_requested` flag in their step definition. In the case where an IPA returns with a different version after reboot, deployment will be terminated.

Post-deploy step hooks will be supported via a `@post_deploy_step_hook` decorator, for example to set a nodes RAID configuration field.

The IPA ramdisk will be modified to add a new `deploy` extension. This will be very similar to the existing `clean` extension. Hardware managers should implement a `get_deploy_steps` method that should work in a similar way to the existing `get_clean_steps` method.

Alternatives

- Deny execution of out-of-band deploy steps before IPA has booted. See Observations for details.
- Allow in-band steps for boot from volume. These could be made available via an optional deploy step that boots up the node on the provisioning network.

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

RPC API impact

None

Driver API impact

There will be no changes to the driver API, but there will be changes to the `AgentDeployMixin` class used by all in-tree drivers and potentially out-of-tree drivers. These changes will be backwards compatible, with a transition period for out-of-tree drivers to switch to the new decomposed step model (advertised by returning `True` from `has_decomposed_deploy_steps`).

Nova driver impact

None

Ramdisk impact

Changes to the IPA ramdisk are discussed above. Backward compatibility will be provided by ignoring a missing `deploy` extension.

Security impact

In-band deploy steps will have additional access to the node by the nature of executing directly on it. These steps and the IPA ramdisk are under the control of the operator, who will need to take action to ensure that they do not introduce any security issues.

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

None

Developer impact

None

Implementation

Assignee(s)

Primary assignees:
Mark Goddard (mgoddard) Arne Wiebalck (arne_wiebalck)

Work Items

- Decompose core deploy step
- Advertise & execute in-band steps via IPA
- Collect & execute in-band steps from IPA
- Update documentation

Dependencies

- Deploy steps (implemented)
- Deploy templates (implemented)

Testing

Ideally tempest tests will be added to cover execution of in-band deploy steps. Software RAID configuration is a reasonable candidate for this as the resulting configuration could be verified.

Upgrades and Backwards Compatibility

This has been discussed elsewhere.

Documentation Impact

The deploy steps documentation will be updated, in particular covering the new flow and the required priorities of user steps.

References

None

5.15.145 In-band RAID configuration using agent ramdisk

https://bugs.launchpad.net/ironic/+bug/1526398

This spec proposes to implement a RAID configuration interface using Ironic Python Agent (IPA). The drivers `agent_ipmitool`, `agent_pyghmi`, `agent_ssh`, `agent_vbox` and `agent_ilo` drivers will make use of this new implementation of `RAIDInterface`.

Problem description

Currently there is no way in Ironic to do RAID configuration for servers using in-band mechanism.

Proposed change

This spec proposes to implement in-band RAID configuration using IPA. It proposes to implement the `RAIDInterface` as mentioned in the parent spec [1]. The implementation will be named `AgentRAID`. The main job of the implementation will be to invoke the corresponding RAID operation methods on agent ramdisk. Interested vendors will implement these methods in Ironic Python Agent using hardware managers.

Following are the changes required:

- The following methods will be implemented as part of `AgentRAID`:
  - `create_configuration` - This will create the RAID configuration on the bare metal node. The following are the steps:
    - Uses `clean.execute_clean_step` command in Ironic Python Agent ramdisk to invoke the `create_configuration` step of `raid` interface.
  - `delete_configuration` - This will delete the RAID configuration on the bare metal node. The following are the steps:
    - Uses `clean.execute_clean_step` command in Ironic Python Agent ramdisk to invoke the `delete_configuration` step of `raid` interface.
- RAID configuration will be limited to zapping only at first by hardcoding its clean step priority to be 0. Well consider interaction with cleaning mechanism later. This allows us to make `target_raid_config` mandatory for the new clean steps.
- When the agent ramdisk is running an in-band clean step, the conductor gets the status of the last in-band clean step on every heartbeat. When an in-band clean step completes, the conductor resumes the cleaning and goes on to the next clean step if any. A new mechanism - the `agent_base_vendor.post_clean_step_hook` decorator, will be added. This allows a driver implementer to specify a function to be invoked after successful completion of an in-band clean step (and before the next clean step is started). The decorated function would take two arguments: the task and the command status (of the clean step) returned by the agent ramdisk.

  For example:

```
@agent_base_vendor.post_clean_step_hook(
    interface='raid', step='create_configuration')
def _create_configuration_final(task, command):
```

- A method `agent._create_configuration_final` will be added as a post clean step hook for `raid.create_configuration`. This method will call `update_raid_info` with the actual RAID configuration returned from the agent ramdisk.
- A method `agent._delete_configuration_final` will be added as a post clean step hook for `raid.delete_configuration`. This will set `node.raid_config` to `None`. Note that `target_raid_config` will be left intact, and will be reused by future zapping calls.
- It is possible to have a hardware manager that does software RAID configuration, but it goes beyond the scope of this spec, as it requires RAID configuration to be run as a clean step after disk erasure.

Alternatives

Some bare metal servers do not support out-of-band RAID configuration. They support only in-band raid configuration. I dont see any other alternatives other than making use of a ramdisk to do this.

We could provide an option to enable RAID as part of cleaning. However, this will make `target_raid_config` mandatory for all nodes managed by a given conductor. Well have to reconsider it later.

Data model impact

None.

State Machine Impact

None.

REST API impact

None.

Client (CLI) impact

None.

RPC API impact

None.

Driver API impact

None.

Nova driver impact

None.

Ramdisk impact

N/A

Security impact

None.

Other end user impact

None.

Scalability impact

None.

Performance Impact

None.

Other deployer impact

None.

Developer impact

Other hardware vendors developing drivers for OpenStack can use Ironic Python Agent for in-band RAID configuration. They can add their own hardware manager implementing the method and get the RAID configuration done.

Implementation

Assignee(s)

rameshg87

Work Items

- Implement the mechanism for post clean step hook.
- Implement `AgentRAID`

Dependencies

- Implement Zapping States - https://review.opendev.org/140826

Testing

Unit tests will be added.

Upgrades and Backwards Compatibility

None.

Documentation Impact

None. Most of the RAID configuration details in Ironic are covered in the parent spec. If anything is required in addition, respective vendors making use of `AgentRAID` will need to document it.

References

[1] http://specs.openstack.org/openstack/ironic-specs/specs/approved/ironic-generic-raid-interface.html

5.15.146 Add Inspect Wait State

https://bugs.launchpad.net/ironic/+bug/1725211

The spec proposes adding a new `inspect wait` state to ironic state machine.

Problem description

The in-band inspection is an asynchronous process[1] that isnt currently handled through a waiting state. This is a discrepancy from the rest of the asynchronous **ironic** states and may comprise a problem for future features such as aborting the introspection[2] or merging **ironic** and **ironic-inspector**[3];

Proposed change

Lets therefore have a new passive state in the **ironic** state machine, the `inspect wait` state.

For asynchronous inspection like ironic inspector driver, ironic conductor will move node from `manageable` to `inspecting` state when an inspect request is issued, then the ironic conductor moves node to `inspect wait` state if `InspectInterface.inspect_hardware` returns `INSPECTING` or `INSPECTWAIT`. The behavior of returning `INSPECTING` will be deprecated and logged as a warning. After deprecation period, returning `INSPECTING` will cause node be moved to `inspect failed` state.

Add a new option `[conductor]inspect_wait_timeout` to guard the `inspect wait` state, the default value is 1800 seconds as same as `[conductor]inspect_timeout`. If the hardware inspection is timed out in the state of `inspect wait`, node will be moved from `inspect wait` to `inspect failed`.

The existing `[conductor]inspect_timeout` will be deprecated.

The `inspect wait` state will be set as an allowed state when updating ironic node, port and portgroup.

As ironic-inspector checks node provision state before starting inspection, the `inspect wait` state needs to be added to ironic-inspector as a valid state.

Alternatives

There are no alternatives to this feature.

Data model impact

None

State Machine Impact

A new unstable state `inspect wait` will be added to ironic state machine.

Following state transitions will be added:

1. `inspecting` to `inspect wait` with event `wait`.
2. `inspect wait` to `inspect failed` with event `fail`.
3. `inspect wait` to `manageable` with event `done`.

[1] https://docs.openstack.org/ironic-inspector/pike/user/http-api.html#start-introspection
[2] https://review.opendev.org/#/c/482867/16/specs/approved/inspection-abort.rst
[3] https://etherpad.openstack.org/p/inspector-queens-virtual-ptg

REST API impact

API microversion will be bumped to hide the new `inspect wait` state to clients with older microversion, this will be done in the `update_state_in_older_versions`.

Node related API endpoints will be affected:

- POST /v1/nodes
- GET /v1/nodes
- GET /v1/nodes/detail
- GET /v1/nodes/{node_ident}
- PATCH /v1/nodes/{node_ident}

For clients with older microversion, the provision state of `inspect wait` will be changed to `inspecting`, there is no other impact to API behaviors.

Client (CLI) impact

As the compatibility is handled at ironic API, CLI is not affected.

ironic CLI

None

openstack baremetal CLI

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

None

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

Add a new option `[conductor]inspect_wait_timeout` to guard timeout of state `inspecting`, defaults to 1800 seconds.

Developer impact

This feature has no impact on synchronous inspection, that includes most of OOB drivers. For in-band inspection, the new state has to be considered.

After this spec is implemented, drivers based on asynchronous inspection have to be changed accordingly, that includes in-band inspection and out-of-band inspection (if there is any).

`OneViewInspect` in the tree is implemented based on ironic inspector interface, its state transition from `inspecting` to `inspect wait` is handled by ironic inspector, but `inspect wait` state needs to be added to status checking.

Implementation

Assignee(s)

Primary assignee:
kaifeng

Other contributors:
vetrisko

Work Items

1. Add `inspect wait` state and state transitions to ironic state machine.
2. Apply state change in the `_check_status` of ironic inspector and `OneViewInspect` driver.
3. Add new option `inspect_wait_timeout`, and deprecate `inspect_timeout`.
4. Handle timeout of state `inspect wait` in the conductor periodic task `_check_inspect_timeouts`, allow updating node, port and portgroup when node is in the `inspect wait` state.
5. Handle API microversion compatibility.
6. Add `inspect wait` to ironic-inspector as a valid state.
7. Update documents, see *Documentation Impact* for details.

Dependencies

None

Testing

Unit tests will be added, API change will be covered by tempest tests.

Upgrades and Backwards Compatibility

The API backwards compatibility is guarded by microvision.

Documentation Impact

The state diagram will be automatically generated from source. Update ironic states document to address the new state, and the semantic change of current `inspecting` state.

References

5.15.147 Support baremetal inspection abort

https://bugs.launchpad.net/ironic/+bug/1703089

This spec aims to support aborting node inspection from ironic API. A dependency of `inspect wait` state in[1] is required for this spec to continue.

Problem description

Currently, we cant abort the process of node inspection from ironic API. When a node is not properly setup under inspection network, admins can only wait it to fail after specified timeout, or abort the inspection process from ironic inspector API/CLI (if the in-band inspect interface `inspector` is in use).

Although the inspection state will be synchronized to ironic by periodic task, its not consistent for an operation started from ironic, then stopped by inspector, furthermore, it creates a little delay of time. Node state is inconsistent between ironic and inspector until next state synchronization. The default time interval for ironic-inspector state synchronization is 60 seconds, it may vary depending on user configuration.

Proposed change

Add state transition of `inspect wait` to `inspect failed` to state machine, add support to ironic to allow the verb `abort` can be requested when node in `inspect wait` state.

Add a method named `abort` into `InspectInterface`, so that inspect interface can provide implementation to support inspection abort. The default behavior is to raise an `UnsupportedDriverExtension` exception. Implement the abort operation for `inspector` inspect interface.

When an abort operation is requested from ironic API, and the node in the state of `inspect wait`, ironic calls `abort` method from inspect interface of driver API, and moves node state to `inspect failed` if the method executed successfully.

Note that, the abort request to ironic-inspector is asynchronous, ironic will move node to `inspect failed` once the request is accepted (202), disregard if the operation at ironic-inspector is performed successfully. This reduces the design complexity for this feature by handling failure at the side of ironic-inspector.

[1] https://specs.openstack.org/openstack/ironic-specs/specs/approved/inspect-wait-state.html

From the point of view of ironic-inspector, every inspect request will refresh local cache for the node, it assures that node state is in sync when starting node inspection. However, inconsistent node state do exist if abort request is accepted but not performed successfully at ironic-inspector. This inconsistency will be eliminated by ironic-inspector node cache clean up when timeout is reached.

Involved changes are:

- Add a method named `abort()` to base inspect interface (InspectInterface).
- Implement `abort()` for `inspector` inspect interface.
- Implement the logic for ironic handling the verb `abort` when provisioning state is `inspect wait`.

Alternatives

- Wait for `inspect fail` after specified timeout value.
- Request through ironic-inspector api or `openstack baremetal introspection abort` command. Be aware that its only viable when using ironic inspector as inspect interface. Other inspect interfaces like out-of-band inspection may have different approach to achieve the same goal, that is beyond the scope of this spec.

Data model impact

None

State Machine Impact

Add a state transition of `inspect wait` to `inspect failed` with event `abort` to ironic state machine.

REST API impact

Modify provision state API to support the transition described in this spec. API microversion will be bumped. For clients with earlier microversion, the verb `abort` is not allowed when a node is in `inspect wait` state.

- PUT /v1/nodes/{node_ident}/states/provision
  - The same JSON Schema is used to `abort` a node in `inspecting` state:

```
{
  "target": "abort"
}
```

  - For client with earlier microversion, 406 (Not Acceptable) is returned
  - For client with supported microversion
    - 202 (Accepted) is returned if request accepted
    - 400 (Bad Request) is returned if current inspect interface does not support abort

Client (CLI) impact

ironic CLI

None

openstack baremetal CLI

None

RPC API impact

None

Driver API impact

A new method `abort` will be added to `InspectInterface` in base.py, the default behavior is to raise the exception `UnsupportedDriverExtension`:

```
def abort(self, task):
    raise exception.UnsupportedDriverExtension(
        driver=task.node.inspect_interface,
        extension='abort')
```

Nova driver impact

None

Ramdisk impact

None

Security impact

None

Other end user impact

None

Scalability impact

For multiple nodes under inspection in a notable scale, it will reduce a little time costs in case of inspection retry.

Performance Impact

None

Other deployer impact

Deployers can abort hardware introspection through ironic API/CLI, besides the inspector API/CLI, for nodes using inspector as the (in-band) inspection interface.

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
kaifeng

Work Items

- Add transition of `inspect wait` to `inspect failed` via `abort`.
- Add a new method `abort()` to the base inspect interface.
- Add the abort implementation to ironic `inspector.Inspector`.
- Implement the abort logic in ironic conductor.

Dependencies

None

Testing

Tempest test will be added to test the REST API change.

Upgrades and Backwards Compatibility

API will be bumped for backward compatibility. Client requests with microversion before this feature will be treated identically.

Documentation Impact

Related documents and state machine diagram will be updated accordingly.

References

5.15.148 Migrate inspection rules from Inspector

https://storyboard.openstack.org/#!/story/2010275

This specification finishes the work started in /approved/merge-inspector by migrating the inspection rules API.

Please see the *Glossary* for the concepts required to understand this specification.

Problem description

Inspection is a pretty opinionated process. A few areas often require site-specific customizations:

- Auto-discovery. For example, credentials may be populated for new nodes following a certain pattern or by fetching them from a CMDB.
- Validation logic. Some operators want to fail inspection if the nodes do not fulfill certain criteria. In the context of auto-discovery, such a validation may be used to prevent unexpected machines from getting enrolled.

These requests can be covered by inspection hooks. But, as explained in *alternatives*, writing and deploying hooks can be too inflexible.

Proposed change

Migrate a streamlined version of introspection rules from Inspector.

These useful features are added on top of what Inspector provides:

Built-in rules
Allow operators to load rules from a YAML file. These rules will be always present, wont be stored in the database and wont be deletable. Such rules are both an easier way to write hooks and a replacement for awkward Inspectors configuration options such as `[discovery]enabled_bmc_address_version`.

Phases
In Inspector, rules are always run at the end of processing. Well add a new field `phase` to rules with values:

- `early` - run before any other processing, even before auto-discovery and lookup. Such rules will not have access to the node object.
- `preprocess` - run after the `preprocess` phase of all inspection hooks, but before the main phase.
- `main` (the default) - run after all inspection hooks.

Updating rules
Inspector does not provide API for updating rules. There is no reason for that, and well add `PATCH` support for them.

Sensitive rules
Conditions and actions of the rules may contain sensitive information, e.g. BMC information. If a rule is marked as sensitive, its actions and conditions will not be returned as part of the `GET` request response. It will not be possible to make a sensitive rule not sensitive.

Error messages resulting from running sensitive rules will also be a bit terse to avoid accidentally disclosing sensitive information.

Priorities
In Inspector, rules are always run in the creation order. This is obviously inconvenient, so in Ironic well add priorities to them. Priorities between 0 and 9999 can be used by all rules, negative value and values above 10000 are reserved for built-in rules. The default priority is 0. Rules within the same priority are still run in the creation order for compatibility.

Database storage
Currently, Inspector spreads each rule into three tables (rules, conditions and actions). This may be more correct from the database design perspective, but is actually inconvenient to work with, since

conditions and actions are never accessed outside of a Rule context. Nor are rules ever accessed without their conditions and actions. This specification buts them as JSON fields inside the Rule table.

Consistent arguments for conditions and actions
A condition has a `field` attribute that is special-cased to be a field of either the node or the inventory. This specification changes both to structure with one operation and several arguments, see *data model impact*.

Alternatives

- Use the inspection hooks mechanism. Hooks are less flexible since they require Python code to be installed alongside Ironic and the service to be restarted on any change. The former is especially problematic for container-based deployments.
- Radically change the rules DSL to something less awkward, e.g. Ansible-like miniscript. While I still want to do it eventually, I think such an undertaking will increate the scope of this already large work too much. With API versioning in place, we can always change the language under the hood.
- Allow API users to upload Python code. No comments.
- Seriously, though, write the rules in Lua or any other grown-up embedded language. I havent researched this option enough. Maybe its the way to go? Will deployers mind a new C dependency (on liblua or LuaJIT)?
- Copy inspection rules verbatim, no removals, no additions. I dont see why not make small improvements for better long-term maintenance. Some of the additions are related to security.
- Make inspection rules into a service separate from Ironic. Defeats the purpose of the Inspector merger. For example, no access to the node database means less efficient operations.
- Do not migrate inspection rules at all. They do bring a bit of complexity, but also proved a helpful instrument for operations. CERN uses them, which is my benchmark for usefulness among advanced operators.

Data model impact

Adapted from Inspector with the additions described in *Proposed change*:

```
class Rule(Base):
    uuid = Column(String(36), primary_key=True)
    created_at = Column(DateTime, nullable=False)
    updated_at = Column(DateTime, nullable=True)
    priority = Column(Integer, default=0)
    description = Column(String(255), nullable=True)
    scope = Column(String(255), nullable=True)  # indexed
    sensitive = Column(Boolean, default=False)
    phase = Column(String(16), nullable=True)  # indexed
    conditions = Column(db_types.JsonEncodedList(mysql_as_long=True))
    actions = Column(db_types.JsonEncodedList(mysql_as_long=True))
```

Conditions and actions

In this specification, both conditions and actions have the same base structure:

- `op` - operation: either boolean (conditions) or an action (actions).
- `args` - a list (in the sense of Python `*args`) or a dict (in the sense of Python `**kwargs`) with arguments.

The special attributes for actions in Inspector get a different form:

- Instead of `invert`: put an exclamation mark (with an optional space) before the `op`, e.g. `eq` - `!eq`.
- Instead of just `multiple`, support an Ansible-style `loop` field. On actions, several actions are run. On conditions, the `multiple` field defines how to join the results. Same as in Inspector:

  any (the default)
  : require any to match

  all
  : require all to match

  first
  : effectively, short-circuits the loop after the first iteration

  last
  : effectively, only runs the last iteration of the loop.

Variable interpolation

String arguments are processed by Python formatting with `node`, `ports`, `port_groups`, `inventory` and `plugin_data` objects available, e.g. `{node.driver_info[ipmi_address]}`, `{inventory[interfaces][0][mac_address]}`.

When running in the early phase, only `inventory` and `plugin_data` are available.

The `node` is actually a proxy mapping taking into account the `mask_secrets` option (described in *other deployer impact*).

If a value is a string surrounded by single curly brackets `{` and `}` (no unformatted text), well evaluate what is inside and avoid converting it into a string. This way, lists and dictionaries can be passed to actions and `loop`. This behavior will likely be implemented by hooking into the Formatter class.

Available conditions

Unlike in Inspector, a list of conditions will be built into Ironic:

`is-true(value)`
: Check if value evaluates to boolean True. On top of actual booleans, non-zero numbers and strings yes, true (in any case) are evaluated to True.

`is-false(value)`
: Check if value evaluates to boolean False. On top of actual booleans, zero `None` and strings no, false (in any case) are evaluated to False.

> **Note**
>
> These conditions can both be false for some values (e.g. random strings). This is intentional.

is-none(value)
Check if value is None.

is-empty(value)
Check if value is None or an empty string, list or a dictionary.

eq/lt/gt(*values, *, force_strings=False)
Check if all values are equal/less than/greater than. If `force_strings`, all values will be converted to strings first.

> **Note**
>
> Inspector has `ne`, `le` and `ge`, which can be implemented via `!eq`, `!gt` and `!lt` instead.

in-net(address, subnet)
Check if the given address is in the provided subnet.

contains(value, regex)
Check if the value contains the given regular expression.

matches(value, regex)
Check if the value fully matches the given regular expression.

one-of(value, values)
Check if the value is in the provided list. Similar to `contains`, but also works for non-string values. Equivalent to:

```
- op: eq
  args: [<value>, "{item}"]
  loop: <values>
```

Available actions

Similar to Inspector, actions will be plugins from the entry point `ironic.inspection_rules.actions`. Coming with Ironic are:

fail(msg)
Fail inspection with the given message.

set-plugin-data(path, value)
Set a value in the plugin data.

extend-plugin-data(path, value, *, unique=False)
Treat a value in the plugin data as a list, append to it. If `unique` is True, do not append if the item exists.

unset-plugin-data(path)
Unset a value in the plugin data.

log(msg, level="info")
Write the message to the Ironic logs.

The following actions are not available in the `early` phase:

set-attribute(path, value)
Set the given path (in the sense of JSON patch) to the value.

extend-attribute(path, value, *, unique=False)
Treat the given path as a list, append to it.

del-attribute(path)
Unset the given path. Fails on invalid node attributes, but does not fail on missing subdict fields.

set-port-attribute(port_id, path, value)
Set value on the port identified by a MAC or a UUID.

extend-port-attribute(port_id, path, value, *, unique=False)
Treat the given path on the port as a list, append to it.

del-port-attribute(port_id, path)
Unset value on the port identified by a MAC or a UUID.

Note

Here *path* is a path in the sense of a JSON patch used by Ironic API.

Examples

Partly taked from the Inspector docs, using YAML format.

```
- description: Initialize freshly discovered nodes
  sensitive: true
  conditions:
    - op: is-true
      args: ["{node.auto_discovered}"]
    - op: "!is-empty"
      args: ["{plugin_data[bmc_address}"]
  actions:
    - op: set-attribute
      args: ["/driver", "ipmi"]
    - op: set-attribute
      args: ["/driver_info/ipmi_address", "{plugin_data[bmc_address]}"]
    - op: set-attribute
      args: ["/driver_info/ipmi_username", "admin"]
    - op: set-attribute
      args: ["/driver_info/ipmi_password", "pa$$w0rd"]
```

Note

The `plugin_data[bmc_address]` field is a side-effect of the `validate_interfaces` hook.

```
- description: Initialize Dell nodes using IPv6
  sensitive: true
  conditions:
    - op: is-true
      args: ["{node.auto_discovered}"]
    - op: contains
      args: ["{inventory[system_vendor][manufacturer]}", "(?i)dell"]
```

(continues on next page)

(continued from previous page)

```
  actions:
    - op: set-attribute
      args: ["/driver", "idrac"]
    - op: set-attribute
      args: ["/driver_info/redfish_address", "https://{inventory[bmc_
↪v6address]}"]
    - op: set-attribute
      args: ["/driver_info/redfish_username", "root"]
    - op: set-attribute
      args: ["/driver_info/redfish_password", "calvin"]
```

State Machine Impact

None (rules are running in the `INSPECTING` state)

REST API impact

Migrate the API mostly verbatim, changing the prefix to `inspection_rules`, adding `PATCH` and more options for listing:

`POST /v1/inspection_rules`
Create an inspection rule. The request body is the representation of the rule. All fields, except for `built_in` can be set on creation. Only `actions` are required (rules with empty conditions run unconditionally).

Returns HTTP 400 on invalid input.

`GET /v1/inspection_rules/<uuid>`
Return one inspection rule. The output fields mostly repeat the database fields, adding a boolean `built_in` field.

For sensitive rules, `null` is returned instead of `conditions` and `actions`.

Returns HTTP 404 if the rule is not found.

`GET /v1/inspection_rules[?detail=true/false&scope=...&phase=...]`
List all inspection rules. If `detail` is `false` or omitted, conditions and actions are not returned. Filtering by scope and phase is possible.

Returns HTTP 400 on invalid input.

`PATCH /v1/inspection_rules/<uuid>`
Update one rule and return it. Sensitive rules can be updated, but the result does not contain conditions or actions in any case.

Returns HTTP 404 if the rule is not found.

Returns HTTP 400 if the input is invalid, e.g. trying to modify `built_in`, change `sensitive` to `false` or set priority outside of the allowed range (0 to 9999).

`DELETE /v1/inspection_rules/<uuid>`
Remove one rule.

Returns HTTP 404 if the rule is not found.

Returns HTTP 400 if the rule is built-in.

DELETE /v1/inspection_rules
: Remove all rules except for built-in ones.

Client (CLI) impact

openstack baremetal CLI

Inspection rules CRUD, adapted from the Introspection Rules CLI, simply replacing *introspection* with *inspection*:

```
$ openstack baremetal inspection rule import <file>
$ openstack baremetal inspection rule list [--long]
$ openstack baremetal inspection rule get <rule ID>
$ openstack baremetal inspection rule delete <rule ID>
```

The mass-deletion command is changed for clarity:

```
$ # Inspector version:
$ openstack baremetal introspection rule purge
$ # New version:
$ openstack baremetal inspection rule delete --all
```

Updating will be possible:

```
$ openstack baremetal inspection rule set <rule ID> \
      [--actions '<JSON>'] [--conditions '<JSON>'] \
      [--sensitive] [--scope '<scope>'] [--phase 'early|preprocess|main'] \
      [--uuid '<uuid>'] [--description '<description>']
$ openstack baremetal inspection rule unset <rule ID> \
      [--conditions] [--scope] [--description]
```

Also adding a way to create from fields instead of one JSON:

```
$ openstack baremetal inspection rule create \
      --actions '<JSON>' [--conditions '<JSON>'] \
      [--sensitive] [--scope '<scope>'] [--phase 'early|preprocess|main'] \
      [--uuid '<uuid>'] [--description '<description>']
```

openstacksdk

The baremetal module will be updated with the standard CRUD plus mass-deletion:

```
def inspection_rules(details=False): pass
def get_inspection_rule(rule): pass
def patch_inspection_rule(rule, patch): pass
def update_inspection_rule(rule, **fields): pass
def delete_inspection_rule(rule, ignore_missing=True):
def delete_all_inspection_rules(): pass
```

RPC API impact

None

Driver API impact

No driver impact. Operators may opt for running inspection rules on nodes with all inspect interfaces, including out-of-band ones.

Nova driver impact

None

Ramdisk impact

None

Security impact

Inspection rules have access to all node and inventory data. Thus, they should be restricted to admins only.

Other end user impact

None

Scalability impact

None

Performance Impact

Having a lot of inspection rules will make inspection longer. But it should not affect the rest of the system.

Other deployer impact

The new section `[inspection_rules]` will have these options:

`built_in`
: An optional path to a YAML file with built-in inspection rules. Loaded on service start and thus not modifiable via SIGHUP.

`default_scope`
: The default value for `scope` for all rules where this field is not set (excluding built-in ones).

`mask_secrets`
: Whether to mask secrets in the node information passed to the rules:

  - `always` (the default) - always remove things like BMC passwords.
  - `never` - never mask anything, pass full node objects to all rules.
  - `sensitive` - allow secrets for rules marked as `sensitive`.

supported_interfaces
: A regular expression to match *inspect interfaces* that run inspection rules. Defaults to `^(agent|inspector)$` to limit the rules to only in-band implementations. Can be set to `.*` to also run on all nodes.

One option will be added to the `[auto_discovery]` section:

inspection_scope
: The default value of inspection scope for nodes enrolled via auto-discovery. Simplifies targeting such nodes with inspection rules.

Developer impact

Actions are provided via plugins with entry points in the `ironic.inspection_rules.actions` namespace:

```
class InspectionRuleActionBase(metaclass=abc.ABCMeta):
    """Abstract base class for rule action plugins."""

    formatted_params = []
    """List of params to be formatted with python format."""

    supports_early = False
    """Whether the action is supported in the early phase."""

    def call_early(self, rule, *args, **kwargs):
        """Run action in the early phase."""
        raise NotImplementedError

    @abc.abstractmethod
    def __call__(self, task, rule, *args, **kwargs):
        """Run action on successful rule match."""
```

Note

The interface in Inspector supports several additional validation features. I hope to derive the valid arguments from method signatures instead.

Implementation

Assignee(s)

Primary assignee:
: Dmitry Tantsur (IRC: dtantsur, dtantsur@protonmail.com)

Other contributors:
: TBD

Work Items

See the RFE.

Dependencies

- /approved/merge-inspector

Testing

- Add functional tests exercising inspection rules CRUD actions.
- Update the in-band inspection job to have a simple rule that we can verify is run (e.g. it sets something in the nodes extra).

Upgrades and Backwards Compatibility

Existing rules will not be automatically migrated from Inspector to Ironic since the conversion may not be always trivial (e.g. around variable interpolation or loops).

Documentation Impact

- API reference will be updated.
- User guide will be migrated from Inspector with a couple of real-life *examples*.

References

5.15.149 Boot and network management for in-band inspection

https://storyboard.openstack.org/#!/story/1584830 https://storyboard.openstack.org/#!/story/1528920

This spec suggests making Ironic Inspector play well with the tenant network separation and non-PXE boot interfaces.

Problem description

With the `neutron` network interface nodes are no longer constantly connected to the provisioning network. We need to connect them manually before in-band inspection, which is inconvenient and error-prone.

This change covers integration with both boot and network interfaces.

Proposed change

The proposed flow will work as follows:

1. Inspection with the `inspector` inspect interface is started via the API.
2. The `inspector` inspect interface:
    1. Calls `task.driver.network.validate_inspection`.

       If it raises `UnsupportedDriverExtension`, fall back to the code path.
    2. Calls `task.driver.boot.validate_inspection`.

       If it raises `UnsupportedDriverExtension`, fall back to the code path.

3. Calls `task.driver.network.add_inspection_network`. It creates a port on the `inspection_network`.
4. Calls `task.driver.boot.prepare_ramdisk` providing kernel parameters from the option `[inspector]extra_kernel_params`.
5. Calls the ironic-inspector introspection API with `manage_boot=False`.
6. Powers on the machine via `task.driver.power`.

3. Now inspection proceeds as previously.

Boot and network interfaces

- Add a new call `validate_inspection`. It will be implemented the same way as `validate_rescue`, but instead of raising `MissingParameterValue` on absent parameters it will raise `UnsupportedDriverExtension` to indicate fall back to the old approach.
  - Implement `validate_inspection` for the PXE and iPXE boot interfaces.
- Add a new `driver_info` parameter `driver_info[inspection_network]` and a new configuration option `[neutron]inspection_network`.
- Extend the `NetworkInterface` to provide `add_inspection_network`, `remove_inspection_network` and `validate_inspection` similarly to rescue networks. However, `validate_inspection` will raise `UnsupportedDriverExtension` if the inspection network is not specified.

Inspector inspect interface

Modify the `Inspector` inspect interface to follow the flow outlined above.

- Call `boot.validate_inspection` and `network.validate_inspection` in the beginning of the introspection process. If either raises `UnsupportedDriverExtension`, follow the same procedure as previously.
- Call `network.add_inspection_network` before and `network.remove_inspection_network` after inspection.
- Add a new `driver_info` parameter `driver_info[inspector_extra_kernel_params]` and a new configuration option `[inspector]extra_kernel_params`.
- Call `boot.prepare_ramdisk` before introspection, providing the ironic-inspector URL (fetched from the service catalog) and `extra_kernel_params` to the `ramdisk_params` argument. Call `boot.cleanup_ramdisk` afterwards.
- Call ironic-inspector passing `manage_boot=False`.

Inspecting ports

Currently Ironic Inspector does not require ports, port groups or local link information to be present to conduct inspection. However, to use network flipping we will need this information, which can be:

- entered manually by an operator (using out-of-band inspection if possible) OR
- inspected initially with a node manually put on the right network.

Alternatives

- Do not support network separation.
- Expose the network and boot interfaces in Ironic API and make Inspector use it.

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

ironic CLI

None

openstack baremetal CLI

None

RPC API impact

None

Driver API impact

Extend the `NetworkInterface` with:

```
def validate_inspection(self, task):
    """Validates the network interface for inspection operation.

    :param task: A TaskManager instance.
    :raises: InvalidParameterValue, if the network interface configuration
        is invalid.
    :raises: MissingParameterValue, if some parameters are missing.
    """
    raise exception.UnsupportedDriverExtension(
        driver=task.node.driver, extension='validate_inspection')

def add_inspection_network(self, task):
    """Add the inspection network to a node.

    :param task: A TaskManager instance.
    :raises: NetworkError
```

(continues on next page)

(continued from previous page)

```
    """
    pass

def remove_inspection_network(self, task):
    """Remove the inspection network from a node.

    :param task: A TaskManager instance.
    """
    pass
```

Extend the `BootInterface` with:

```
def validate_inspection(self, task):
    """Validate that the node has required properties for inspection.

    :param task: A TaskManager instance with the node being checked
    :raises: MissingParameterValue if node is missing one or more required
        parameters
    :raises: UnsupportedDriverExtension
    """
    raise exception.UnsupportedDriverExtension(
        driver=task.node.driver, extension='validate_inspection')
```

Nova driver impact

None

Ramdisk impact

None

Security impact

This change will also allow using in-band inspection with tenant network separation increasing security.

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

New configuration options:

- `[neutron]inspection_network` the default inspection network (no default).
- `[inspector]extra_kernel_params` the default kernel parameters to pass to introspection (empty by default).

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
Dmitry Tantsur (lp: divius, irc: dtantsur)

Work Items

1. Add new methods to the network and boot interfaces.
2. Update the `inspector` inspect interface to use them.

Dependencies

None

Testing

Coverage by unit tests.

Upgrades and Backwards Compatibility

The default behavior will not change because the `inspection_network` will be left unpopulated initially. After it gets populated, nodes with ports will follow the new flow for introspection. This feature can be enabled per node by setting `inspection_network` on nodes, not globally.

This work does not anyhow affect introspection that is started using the ironic-inspectors own CLI or API.

Documentation Impact

The Ironic documentation should be updated to explain using network separation with in-band inspection.

References

5.15.150 Interface Attach and Detach API

https://bugs.launchpad.net/ironic/+bug/1582188

We are adding pluggable network interfaces to Ironic. These interfaces will be configurable per Ironic node, this means that different nodes might have different ways of handling their network interfaces. We need a way that different network interfaces can override the virtual network interface (VIF) to physical network interface (PIF) mapping logic currently contained in nova.

Problem description

Currently we use an Ironic port update to assign tenant neutron ports to Ironic ports using their extra field. Doing the mapping this way may not work for a third party network interface implementation. This also ties the ironic nova virt driver to a particular network VIF to PIF mapping implementation. If a third party network interface wants to do something different with network VIF to PIF mapping such as storing the port IDs for dynamic vNIC creation later in the provisioning process or we as the Ironic team want to change an implementation detail we have to submit changes into nova. Additionally if we want to support post-deployment attach and detach of a network VIF then we have to watch for any updates made to the port object and interpret certain changes as certain actions.

Proposed change

To solve this problem I propose to add a new API endpoint,

- POST v1/nodes/<node_id>/vifs
- GET v1/nodes/<node_id>/vifs
- DELETE v1/nodes/<node_id>/vifs/<vif_id>

These API endpoints will take via a POST body, a JSON representation of a generic VIF object. Making it generic allows for non-neutron based implementations to use this API. This VIF object will be passed to new functions in the pluggable network interfaces:

def vif_attach(self, vif):

def vif_detach(self, vif_id):

def vif_list(self):

The network interface can use these functions to handle attaching the VIF to the Ironic node in whichever way it needs to for its implementation. This could be by adding a field to the Ironic port as with the existing implementation, or it might be different for example storing it in a list in the nodes driver_internal_info.

The ironic nova virt driver will be updated to use this new API in the plug_vifs and unplug_vifs functions, unbinding it from the underlying implementation details.

Alternatives

- Continue to use a port update to allow nova to interact with ironic ports and soon portgroups, documenting the vif_port_id etc logic as a defined API for the network interfaces to use.

Data model impact

None

State Machine Impact

None

REST API impact

- GET v1/nodes/<node_id>/vifs
  - Calls to vif_list on the nodes network interface

  - This is a synchronous API endpoint, and the normal ironic rules apply for avoiding implementing tasks that take a long time or are unstable under synchronous endpoints.
  - Method: GET
  - Successful http response: 200
  - Expected response body is a JSON
    * JSON Schema:

```
{
    "title": "VIFS",
    "type": "object",
    "properties": {
        "vifs": {
            "description": "List of VIFs currently attached"
            "type": "array",
            "items": {
                "type": "object",
                "properties": {
                    "id": {
                        "type": "string"
                    }
                },
                "required": ["id"]
            }
        }
    },
    "required": ["vifs"]
}
```

    * Example:

```
{
    "vifs": [
                {
                    "id": "8e6ba175-1c16-4dfa-82b9-dfc12f129170",
                }
    ]
}
```

- POST v1/nodes/<node_id>/vifs
  - Calls vif_attach on the network interface for the node, passing in the json provided
  - This is a synchronous API endpoint, and the normal ironic rules apply for avoiding implementing tasks that take a long time or are unstable under synchronous endpoints.
  - Method: POST
  - Successful http response: 204
  - Expected error response codes:
    * 404, Node not found

        * 400, Request was malformed
        * 409, Conflict between the requested VIF to attach and other VIFs already attached
        * 422, The request was good but unable to attach the VIF for a defined reason, for example: No physical interfaces available to attach too
    - Expected data is a JSON
        * JSON Schema:

```
{
    "title": "Attachment",
    "type": "object",
    "properties": {
        "id": {
            "description": "ID of VIF to attach"
            "type": "string"
        },
    },
    "required": ["id"]
}
```

        * Example:

```
{
  "id": "8e6ba175-1c16-4dfa-82b9-dfc12f129170"
}
```

    - Expected response body is empty
- DELETE v1/nodes/<node_id>/vifs/<vif_id>
    - Calls vif_detach on the network interface for the node, passing in the json provided
    - This is a synchronous API endpoint, and the normal ironic rules apply for avoiding implementing tasks that take a long time or are unstable under synchronous endpoints.
    - Method: DELETE
    - Successful http response: 204
    - Expected error response codes:
        * 404, Node not found
        * 400, Request was malformed
        * 422, The request was good but unable to detach the VIF for a defined reason
    - Expected response body is empty.
- Does the API microversion need to increment? Yes
- Is a corresponding change in the client library and CLI necessary? Yes
- As these are new API entry points they will not affect older clients.

Client (CLI) impact

ironic CLI

- ironic node-vif-list <node_id>
- ironic node-vif-attach <node_id> <vif_id>
- ironic node-vif-detach <node_id> <vif_id>

openstack baremetal CLI

- openstack baremetal node vif list <node_id>
- openstack baremetal node vif attach <node_id> <vif_id>
- openstack baremetal node vif detach <node_id> <vif_id>

RPC API impact

The RPC API will implement:

- vif_attach(self, context, node_id, vif)
- vif_detach(self, context, node_id, vif_id)
- vif_list(self, context, node_id)

Driver API impact

Base network interface will need to be extended with:

```
def vif_list(self, task):
    # TODO(sambetts): Uncomment when vif_port_id in port.extra is removed.
    # raise NotImplemented
    default_vif_list()

def vif_attach(self, task, vif):
    # TODO(sambetts): Uncomment when vif_port_id in port.extra is removed.
    # raise NotImplemented
    default_vif_attach(vif)

def vif_detach(self, task, vif_id):
    # TODO(sambetts): Uncomment when vif_port_id in port.extra is removed.
    # raise NotImplemented
    default_vif_detach(vif_id)
```

Existing flat, neutron and noop network interfaces will need extending to include implementations for these functions.

Flat network driver will need to implement add_provisioning_network to bind the ports that used to be bound by nova.

Nova driver impact

plug/unplug_vifs logic will be replaced by calling attach/detach for every VIF passed into those functions.

nova.virt.driver.IronicDriver.macs_for_instance will be removed because mapping is handled inside Ironic so mac_address assignment must happen at binding later in the process.

nova.virt.driver.IronicDriver.network_binding_host_id will be changed to return None in all cases, so that neutron ports remain unbound until Ironic binds them during deployment.

nova driver will need to include the nova compute host ID in the instance_info so that the ironic flat network interface can use it to update the neutron ports mimicking the existing nova behavior.

The nova driver ironic API version requirement will need to be increased to the version that implements the attach and detach APIs. Operators will need to ensure the version of python-ironicclient they have installed on the nova compute service supports the new APIs.

Ramdisk impact

None

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

None

Developer impact

Developers of network interfaces will need to consider how their network interface wants to handle ports.

Implementation

Assignee(s)

Primary assignee:
sambetts

Work Items

- Add new APIs to Ironic
- Update existing ironic network interfaces to support the new APIs
- Add new APIs to ironic client
- Update nova virt driver to use new APIs via client

Dependencies

- Ironic neutron integration
  - https://specs.openstack.org/openstack/ironic-specs/specs/not-implemented/ironic-ml2-integration.html

Testing

These changes will be tested as part of the normal gate process, as they will be part of the normal ironic deployment workflow.

Upgrades and Backwards Compatibility

Setting vif_port_id via a port/portgroup update will be deprecated in favor of the new APIs. A deprecation message should be issued on a port update if a user is directly setting vif_port_id on a port or portgroup. Code will need to be added to ensure that the network interfaces will still process a vif_port_id set via that method.

As the old vif_port_id field is still supported although deprecated, old nova virt drivers will continue to work using that method.

Ironic must be upgraded before Nova is as newer nova virt drivers wont work with older versions of Ironic which are missing the new APIs.

Out-of-tree network interfaces may not immediately implement the interface_attach and interface_detach methods, so during the period that vif_port_id is a deprecated method, we should also ensure we provide a default implementation of attach and detach, this default implementation should match the behaviour of ironic nova virt driver implementation setting the vif_port_id in port.extra and will be removed when support for vif_port_id in port.extra is removed.

Documentation Impact

New API and its usage needs to be documented.

References

None

5.15.151 ironic-python-agent API Versioning and Negotiation

https://bugs.launchpad.net/ironic/+bug/1602265

It was decided at the ironic newton midcycle that we need some form of API version negotiation between ironic and IPA. This is required to allow ironic to recognise that it is talking to an older ramdisk, and that newer features will not be available.

Problem description

During ironic upgrade its possible that you will end up with node which uses an version of the ironic-python-agent (IPA) older than the version of ironic. In this case newer features that have been added to ironic and IPA will be expected to exist by ironic however because the ramdisk used on a node has not been upgraded yet, those features do not exist in that version of IPA. This leads to failures because ironic tries to use these features during deployment/cleaning and resulting in unexpected responses from IPA.

Proposed change

The proposed change is that when IPA does its heartbeat to ironic it will include an IPA version number that will be cached by ironic in the nodes `driver_internal_info` in the same way as the Agent URL is stored. ironic will then use this version information to gracefully degrade the feature set that it uses from the ironic-python-agent. For example:

```
def do_something_using_IPA(self)
  # Make sure to refresh the node object in case agent_version has changed
  # via a heartbeat since we first fetched the node.
  node = self.node.refresh()
  if node.driver_internal_info.get('agent_version', '0.0.0') >= '1.2':
    do_new_additional_thing_only_supported_in_1.2()
  do_thing_using_IPA()
```

Alternatives

A couple of alternatives exist:

- Require that operators update every nodes ramdisk to the newer version of the ramdisk with the new features before upgrading their ironic installation.
- Handle the cases where IPA doesnt support a particular feature by catching a set of expected errors on a case by case basis and gracefully fall back to another method. However in cases where this occurs it will result in more complex error handling in Ironic and extra API calls to IPA.

Data model impact

New `agent_version` field will be stored in `node.driver_internal_info` on agent heartbeat.

State Machine Impact

In the existing code the cleaning step checks the IPA hardware manager version and if it changes for any reason during the cleaning process, then cleaning is aborted and will need to be restarted. The agent version being added by this spec should be treated in the same way to ensure the most stable environment for cleaning.

REST API impact

A new field `agent_version` will we included in the body of the heartbeat API request alongside the existing `agent_url` field.

Client (CLI) impact

None

ironic CLI

None

openstack baremetal CLI

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

IPA will need updating to include its current version in the `agent_version` field when it makes a heartbeat request.

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

This change will allow for deployers upgrading their ironic installs to upgrade their nodes ironic-python-agent ramdisks asynchronously from the rest of ironic.

Developer impact

Developers of ironic drivers that interact with the ironic-python-agent will need to ensure their code takes into account the agent version that they might be talking to. Making sure that if talking to an IPA that does not support a feature it disables that feature in ironic.

Implementation

Assignee(s)

Primary assignee:
sambetts

Work Items

- Add code to IPA to send its version to ironic in the heartbeat request
- Add code to ironic to accept and store the `agent_version` when it receives a heartbeat
- Add developer documentation on the correct way to add support for an IPA feature in ironic, such that it will gracefully degrade if it is not available.

Dependencies

None

Testing

- ironic grenade tests already test ironic master with the last named release IPA
- Normal ironic/IPA tests will test ironic master + IPA master builds
- Need to add a grenade test to IPA to test last ironic named release + master IPA

Upgrades and Backwards Compatibility

- This spec adds the ability to better support Ironic version N+1 with IPA version N or older as Ironic will gracefully degrade which features it will request if they arent available
- Ironic version N works with IPA version N+1 and should continue to work

Documentation Impact

- Need to add the developer documentation mention in the work items section
- Need to document which versions of IPA are supported with which versions of ironic.

References

- https://etherpad.openstack.org/p/ironic-newton-midcycle

5.15.152 Promote iPXE to separate boot interface

https://bugs.launchpad.net/ironic/+bug/1628069

The iPXE boot should be promoted to a separate boot driver interface. This will simplify the code base and allow to not force global PXE *vs* iPXE conductor setting.

Problem description

Currently setting PXE or iPXE is enforced per-conductor, and taking into account the possibility of node take-over, this choice is effectively global for the whole Ironic installation.

Due to this the current code of `PXEBoot` interface is riddled with constructs of:

```
if CONF.pxe.ixpe_enabled ...
```

Recently added or proposed changes (like `CONF.pxe.ipxe_use_swift` option) make the logic there even more complicated.

Proposed change

It is proposed to implement a separate iPXE boot interface, which will use the new way of serving iPXE boot scripts and boot config files directly from Ironic API as outlined in dynamic iPXE configuration spec.

The new interface will get a separate `[ipxe]` config section, where all `[pxe]ipxe_*` options should be moved following proper deprecation policy for config options. Current iPXE-related behavior of `PXEBoot` interface should be deprecated and eventually removed.

Alternatives

Continue mixing PXE and iPXE in single driver interface and setting iPXE *vs* PXE globally.

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

ironic CLI

None

openstack baremetal CLI

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

None

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

- A new config section `[ipxe]` will be added, with most of the ipxe_* options copied from current `[pxe]` section (with `ipxe_` removed from option names). By the virtue of `oslo_config` library, the new options will be backward compatible with old, deprecated ones, using their values when not set explicitly.
- This change has no immediate effect. Enabling it is a conscious choice of the operator:
  - a driver utilizing this new `iPXEBoot` boot interface must be composed
  - such driver must be assigned to the node

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
pshchelo (Pavlo Shchelokovskyy)

Work Items

- create a new `iPXEBoot` boot interface
  - identify and refactor code that is common for `PXEBoot` and `iPXEBoot` interfaces
- register this driver as entry point for `ironic.hardware.interfaces.boot`
  - add it to list of boot interfaces enabled by default in ironic config (`[DEFAULT]enabled_boot_interfaces` config option)
- add appropriate documentation

Dependencies

- Dynamic iPXE configuration
- Driver composition reform

Testing

No specific coverage seems to be needed apart from enabling a driver that uses this new proposed boot interface at least on one gate job.

Upgrades and Backwards Compatibility

The feature has no immediate effect on existing installation as it must be manually enabled first by enabling the interface and composing an appropriate driver with this boot interface.

Existing drivers do not depend on this feature in any way.

It is also proposed to deprecate and eventually remove iPXE capabilities from the PXEBoot interface.

Chain-loading an iPXE-capable boot-loader will still be supported by iPXEBoot driver to support older/dumber/buggy hardware.

Documentation Impact

Document new driver interface and its usage.

References

5.15.153 iPXE to use Swift Temporary URLs

https://bugs.launchpad.net/ironic/+bug/1526404

This adds support for generating Swift temporary URLs for the deploy ramdisk and kernel when booting with iPXE.

Problem description

Currently the iPXE driver requires an external HTTP server to serve the deploy ramdisk and kernel. When used with Glance, the `ironic-conductor` fetches the images from it and place them under the HTTP root directory, and if a rebalance happens in the hash right the new `ironic-conductor` taking over the node have to do the same thing, fetch the images and cache it locally to be able to manage that node.

Having an external HTTP server should not be required when Glance is used with a Swift backend, with Swift we can generate temporary URLs that can be passed to iPXE to download the images without requiring credentials.

Proposed change

The proposed implementation consists in having the iPXE driver to create a Swift tempurl[0] for the deploy ramdisk and kernel that the node will boot as part of the config generation.

This also proposes adding a boolean configuration option under the `pxe` group called `ipxe_use_swift`. If True this will tell iPXE to not cache the images in the disk and generate the Swift tempurl for the ramdisk and kernel, if False, iPXE will continue to cache the images under the HTTP root directory. Defaults to False.

Note that in order to keep compatibility with Nova behavior, kernel/ramdisk of the user image still have to be cached in case `netboot` is required. Doing otherwise will make it impossible for user to reboot the instance from within when tempurls have expired or the image is deleted from Glance altogether.

Alternatives

Continue to use an external HTTP server and caching the images on the disk.

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

[0] http://docs.openstack.org/kilo/config-reference/content/object-storage-tempurl.html

Ramdisk impact

N/A

Security impact

Theres a positive security impact because the Swift temporary URLs does have an expiration time and the images in the external HTTP server will be available until the instance is destroyed.

Other end user impact

None

Scalability impact

There is a scaling benefit to download directly from Swift since a Swift cluster can be scaled horizontally by adding new nodes.

Performance Impact

None

Other deployer impact

None

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
lucasagomes <lucasagomes@gmail.com>

Other contributors:
pshchelo <shchelokovskyy@gmail.com>

Work Items

- Add the new `ipxe_use_swift` configuration option under the `pxe` group.
- Get the PXE driver to generate the Swift temporary URLs as part of the configuration generation when `ipxe_use_swift` is True.
- Skip caching the image on the disk when `ipxe_use_swift` is True.

Dependencies

None

Testing

Unittests will be added.

Upgrades and Backwards Compatibility

None

Documentation Impact

The iPXE documentation will be updated to reflect the changes made by this spec.

References

5.15.154 New driver interface for RAID configuration

https://bugs.launchpad.net/ironic/+bug/1526400

The proposal presents the work required to create a new driver interface for RAID configuration. It also proposes a method to make RAID configuration available as part of zapping or cleaning.

Note

Even though RAID configuration fits into zapping, it can be used as part of cleaning as well. Zapping and cleaning follow a similar mechanism (zap step is a clean step with priority of 0). It makes sense to do RAID configuration as part of cleaning for software RAID (where secure disk erase will also erase the software RAID configuration on the disk). Its operators choice to decide whether RAID configuration should be part of zapping or cleaning and it will be configurable in the drivers implementing it.

Problem description

- There is no support in Ironic currently to do RAID configuration.
- A specific set of tasks for this requires a separate interface on the drivers. The new RAID interface will allow operators to specify RAID configuration for a node. Different drivers may provide the same interface to the operator for RAID configuration.

Proposed change

- After a node is enrolled and the basic hardware information is available, the operator can define a RAID configuration. This configuration will be applied during zapping or cleaning.
- The operator can convey the RAID configuration information to the Ironic driver through REST APIs or CLI as JSON data. The RAID configuration information will contain the properties for each logical disk and optionally hints to Ironic to find the desired backing physical disks for them.

  The properties can be split into 4 different types:

  1. Mandatory properties - These properties must be specified for each logical disk and have no default values.

     - `size_gb` - Size (Integer) of the logical disk to be created in GiB. `MAX` may be specified if the logical disk should use all of the space available on the backing physical disks. This can be used only when backing physical disks are specified (see below).

  - `raid_level` - RAID level for the logical disk. Ironic will define the supported RAID levels as 0, 1, 2, 5, 6, 1+0, 5+0, 6+0. Drivers may override the values in the `get_logical_disk_properties` method in `RAIDInterface`.

2. Optional properties - These properties have default values and they may be overridden in the specification of any logical disk.

  - `volume_name` - Name of the volume. Should be unique within the Node. If not specified, volume name will be auto-generated.

  - `is_root_volume` - Set to `true` if this is the root volume. Can be used for only one of logical disk. The root device hint will be saved, if the driver is capable of retrieving it. This is `false` by default.

3. Backing physical disk hints - These hints are specified for each logical disk to let Ironic find the desired disks for RAID configuration. This is machine-independent information. This serves the use-case where the operator doesnt want to provide individual details for each bare metal node.

  - `share_physical_disks` - Set to `true` if this logical disk can share physical disks with other logical disks. If this is not provided, drivers will assume it as `false`.

  - `disk_type` - `hdd` or `ssd`. If this is not specified, disk type will not be considered as a criteria to find backing physical disks.

  - `interface_type` - `sata` or `scsi` or `sas`. If this is not specified, interface type will not be considered as a criteria to find backing physical disks.

  - `number_of_physical_disks` - Integer, number of disks to use for the logical disk. Defaulted to minimum number of disks required for the particular RAID level.

  The above mentioned backing physical disk hints are defined by Ironic and every driver has to implement them. The supported values and the default values for the above hints may be overridden by the driver using the `RAIDInterface.get_logical_disk_properties()` method.

  In addition to the above hints, drivers may define their own hints in the `get_logical_disk_properties` method. For more details, refer to the Driver API impact section. The possible use-cases for them might be:

  - Filter disks by particular vendors

  - Filter disks by models

  - Filter disks by firmware versions.

4. Backing physical disks - These are the actual machine-dependent information. This is suitable for environments where the operator wants to automate the selection of physical disks with a 3rd-party tool based on a wider range of attributes (eg. S.M.A.R.T. status, physical location).

  - `controller` - The name of the controller as read by the driver.

  - `physical_disks` - A list of physical disks to use as read by the driver.

  Note

  The values for these properties are hardware dependent.

Note

Only properties from Backing physical disk hints or Backing physical disks should be specified. If both are specified, they should be consistent with each other. If they are not consistent, then the raid configuration will fail (because the appropriate backing physical disks could not be found).

Some examples:

Example 1 (using backing physical disk hints):

```
{
  "logical_disks":
    [
      {
        "size_gb": 50,
        "raid_level": "1+0",
        "disk_type": "hdd",
        "interface_type": "sas",
        "volume_name": "root_volume",
        "is_root_volume": "true"
      },
      {
        "size_gb": 100,
        "number_of_physical_disks": 3,
        "raid_level": "5",
        "disk_type": "hdd",
        "interface_type": "sas"
        "volume_name": "data_volume"
      }
    ]
}
```

Example 2 (using backing physical disks):

```
{
  "logical_disks":
    [
      {
        "size_gb": 50,
        "raid_level": "1+0",
        "controller": "RAID.Integrated.1-1",
        "volume_name": "root_volume",
        "is_root_volume": "true"
        "physical_disks": [
                           "Disk.Bay.0:Encl.Int.0-1:RAID.Integrated.
↪1-1",
                           "Disk.Bay.1:Encl.Int.0-1:RAID.Integrated.
↪1-1"
                          ]
      },
      {
```

(continues on next page)

(continued from previous page)

```
        "size_gb": 100,
        "raid_level": "5",
        "controller": "RAID.Integrated.1-1",
        "volume_name": "data_volume"
        "physical_disks": [
                            "Disk.Bay.2:Encl.Int.0-1:RAID.Integrated.
↪1-1",
                            "Disk.Bay.3:Encl.Int.0-1:RAID.Integrated.
↪1-1",
                            "Disk.Bay.4:Encl.Int.0-1:RAID.Integrated.
↪1-1"
                           ]
      }
    ]
}
```

- The RAID configuration information is stored as JSON in `node.target_raid_config` field. Operator can use the REST API (or CLI) to put a new value here at any time, which is compared to `node.raid_config` during zapping and cleaning, and driver may apply changes only in those stages. Refer REST API Impact section for more details.
- New driver interface called `RAIDInterface` will be provided for RAID configuration for drivers. For more details, refer to the Driver API impact section.
- New methods `create_configuration` and `delete_configuration` in `RAIDInterface` will be available as part of zapping. The operator can choose to call them as part of zap steps. The corresponding zap steps will be `node.raid.create_configuration` and `node.raid.delete_configuration`.
- A new method `update_raid_info` will be available in ironic.common.raid. This method may be used by the drivers implementing RAID support to update the RAID information in the Node database. This will facilitate drivers to do the RAID configuration asynchronously. This method will do the following:
  - Set `node.raid_config` to the value returned by the driver.
  - The root device hint for the root volume will be updated in `node.properties` (as per root device hint) and the size of root volume will be updated in `node.properties.local_gb`. Its up to the driver to choose which root device hint it wants to specify. Furthermore, it isnt even necessary for the driver to choose any root_device_hint.
  - The RAID level of the root volume will be updated as `raid_level` in `node.properties.capabilities`.
- A new REST API will be created for retrieving the properties which may be specified as part of RAID configuration. For details, see the REST API Impact section below.
- REST API will be created to PUT RAID config, and a new REST resource added to retrieve the requested and actual RAID config.

Alternatives

- Operator can change the RAID configuration manually whenever required after putting the node to MANAGEABLE state. But this has to be done for each node.

Data model impact

The following fields in the Node object will be updated:

- A new database field, `node.target_raid_config`, will store the pending RAID configuration to be applied during zapping or cleaning. This will be a JSON dictionary. This field will be read-only.
- A new database field, `node.raid_config`, will store the last applied RAID configuration. This will also contain the timestamp of when this configuration was applied. This will be a JSON dictionary. This field will be read-only.
- `node.properties.local_gb` will be updated after applying RAID configuration to the size of the root volume.
- `node.properties.root_device` will be updated with the root device hint returned by the driver as prescribed in the root device hint spec.
- A new capability `raid_level` will be added in `node.properties.capabilities`. This will contain the RAID level of the root volume.

State Machine Impact

None.

REST API impact

Two new REST API endpoints will be introduced as part of this change.

- To GET the RAID properties that can be defined and their possible values:

```
GET /drivers/<driver>/raid/logical_disk_properties
```

  The operation will return the properties and a textual description of the possible values for each property:

```
{
 'raid_level': 'RAID level for the logical disk. Supported values are
                0, 1, 2, 5, 6, 1+0, 5+0, 6+0. Required.',
 'size_gb': 'Size of the logical disk in GiB. Required.',
 'disk_type': 'Disk Type. Supported values are `hdd` or `sdd`. Optional',
 .
 .
 .
 .
}
```

- To set the target RAID configuration, a user will:

```
PUT /v1/nodes/NNNN/states/raid
```

with a BODY containing the JSON description of the RAID config.

If accepted by the driver, this information will be stored in the `node.target_raid_config` field and exposed in the same manner as the power and provision states. In other words, it may be retrieved either within the detailed view of a `node`, or by either of the following:

```
GET /v1/nodes/NNNN
GET /v1/nodes/NNNN/states
```

Note

It might also make sense to have GET /v1/nodes/NNNN/states/raid, but for maintaining consistency with power and provision, we allow only GET /v1/nodes/NNNN and GET /v1/nodes/NNNN/states.

If the driver doesnt support RAID configuration, then both API calls will return HTTP 400 (Bad Request). Otherwise the API will return HTTP 200 (OK).

Client (CLI) impact

A new option will be available in Ironic CLI for getting the properties which may be specified as part of the RAID configuration:

```
$ ironic node-raid-logical-disk-properties <node-uuid>
```

A new method will be added to set the target RAID properties

RPC API impact

Two new RPC APIs will be created.

- `get_raid_logical_disk_properties` - This method will be called in `GET /drivers/<driver>/raid/logical_disk_properties`.
- `set_target_raid_config` - This method will be called in `PUT /v1/nodes/NNNN/states/raid`.

Driver API impact

A new `RAIDInterface` will be available for the drivers to allow them to implement RAID configuration. It will have the following methods:

- `create_configuration()` - The driver implementation of the method has to read the request RAID configuration from `node.target_raid_config` and create the RAID configuration on the bare metal. The driver implementations should throw error if `node.target_raid_config` is not set. The driver must ensure that `ironic.common.raid.update_raid_info()` is called at the end of the process, in order to update the nodes `raid_config`. The implementation detail is up to the driver depending on the synchronicity/asynchronicity of the operation.

  The `raid_config` will include the following:

  - For each logical disk (in addition to the input passed):
    - `controller` - The name of the controller used for the logical disk as read by the driver.

    * `physical_disks` - A list containing the identifier for the physical disks used for the logical disk as read by the driver.
    * `root_device_hint` - A dictionary containing the root device hint to be used by Ironic to find the disk to which image is to be deployed. Its up to the driver to determine which root device hint it wants to provide.

- A list of all the physical disks on the system with the following details:
    * `controller` - RAID controller for the physical disk.
    * `id` - ID for the physical disk as read the driver
    * `disk_type` - `hdd` or `ssd`
    * `interface_type` - `sas` or `sata` or `scsi`
    * `size_gb`
    * `state` - State field states the current status of the physical disk. It can be one of:
        - `active` if disk is part of an array
        - `ready` if disk is ready to be part of a volume
        - `failed` if disk has encountered some error
        - `hotspare` if disk is hotspare and part of some array
        - `offline` if disk is not available for raid due to some other reason, but not failed
        - `non_raid` if disk is not part of raid and is directly visible

  The above details may be used for backing physical disk hints for later raid configurations.

> **Note**
>
> For a newly enrolled node or a node in which raid configuration was never done, the information about physical disks and controllers can be populated by hardware introspection. This is not in the scope of this spec.

The function definition will be as follows:

```
def create_configuration(task, create_root_volume=False,
                         create_nonroot_volumes=False):
    """Create RAID configuration on the node.

    This method creates the RAID configuration as read from
    node.target_raid_config.  This method
    by default will create all logical disks.

    :param task: TaskManager object containing the node.
    :param create_root_volume: Setting this to False indicates
        not to create root volume that is specified in the node's
        target_raid_config. Default value is True.
    :param create_nonroot_volumes: Setting this to False indicates
        not to create non-root volumes (all except the root volume) in
```

(continues on next page)

(continued from previous page)

```
        the node's target_raid_config.  Default value is True.
    :returns: states.CLEANWAIT if RAID configuration is in progress
        asynchronously or None if it is complete.
    """
```

- `delete_configuration()` - To delete the RAID configuration.

  The function definition will be as follows:

```
def delete_configuration(task):
    """Delete RAID configuration on the node.

    :param task: TaskManager object containing the node.
    :returns: states.CLEANWAIT if deletion is in progress
        asynchronously or None if it is complete.
    """
```

- `validate()` - To validate a RAID configuration. This will be called while validating the driver interfaces. This will read the target RAID configuration from node.properties.target_raid_config and call `validate_raid_config` to validate target RAID configuration.

  The function definition will be as follows:

```
def validate(task):
    """Validates the RAID interface.

    :param task: TaskManager object containing the node.
    :raises: InvalidParameterValue, if RAID configuration is invalid.
    :raises: MissingParameterValue, if RAID configuration has some
        missing parameters.
    """
```

- `validate_raid_config()` - To validate target RAID configuration. This will be called during the RPC call `set_target_raid_config()` to validate target RAID configuration. It will also be called during `validate()`.

  The function definition will be as follows:

```
def validate_raid_config(task, raid_config):
    """Validates the given RAID configuration.

    :param task: TaskManager object containing the node.
    :param raid_config: The target RAID config to validate.
    :raises: InvalidParameterValue, if RAID configuration is invalid.
    """
```

- `get_logical_disk_properties()` - To get the RAID properties that are defined by the driver.

  The function definition will be as follows:

```
def get_logical_disk_properties():
    """Gets the RAID properties defined by the driver.
```

(continues on next page)

(continued from previous page)

```
    :returns: A dictionary of properties and a textual description.
    """
```

After performing the RAID configuration (create or delete), the drivers may call `ironic.common.raid.update_raid_info()` with the `raid_config`. The details about the method has been described above. The definition of the method will look like below:

```
def update_raid_info(node, raid_config):
    "Updates the necessary fields of the node after RAID configuration.

    This method updates the current RAID configuration in
    node.properties.raid_config.  If root device hint was passed,
    it will update node.properties.local_gb, node.properties.root_device_hint
    and node.properties.capabilities['raid_level'].

    :param node: a node object
    :param raid_config: The current RAID configuration on the bare metal
        node.
    """
```

Nova driver impact

None.

Ramdisk impact

N/A

Security impact

None.

Other end user impact

Users from Nova may choose the desired RAID level for the root volume by using compute capabilities. For example:

```
nova flavor-key ironic-test set capabilities:raid_level="1+0"
```

Scalability impact

None.

Performance Impact

RAID configuration may extend the time required for zapping or cleaning on the nodes, but this is important for performance and reliability reasons.

Other deployer impact

Operator can make use of `node.raid.create_configuration` and `node.raid.delete_configuration` as zap or clean tasks for doing RAID management.

Developer impact

Developers may implement the `RAIDInterface` for respective drivers.

Implementation

Assignee(s)

Primary assignee:
rameshg87

Other contributors:
ifarkas

Work Items

- Create REST API endpoints for RAID configuration.
- Create `RAIDInterface` and create a fake implementation of `RAIDInterface`.
- Implement `update_raid_info` in ironic.common.raid.
- Implement Ironic CLI changes.
- Write unit tests.

Dependencies

- Root device hints - http://specs.openstack.org/openstack/ironic-specs/specs/kilo/root-device-hints.html
- Zapping of nodes - https://review.opendev.org/#/c/140826/

Testing

- Unit tests will be added for the code. A fake implementation of the `RAIDInterface` will be provided for testing purpose and this can be run as part of zapping.
- Tempest API coverage will be added, using the fake driver above.
- Each driver is responsible for providing the third party CI for testing the RAID configuration.

Upgrades and Backwards Compatibility

None.

Documentation Impact

Documentation will be provided on how to configure a node for RAID.

References

Other references:

- New Ironic provisioning state machine: http://specs.openstack.org/openstack/ironic-specs/specs/kilo/new-ironic-state-machine.html
- Support Zapping of Nodes: https://review.opendev.org/#/c/140826/

5.15.155 Ironic Neutron Integration

https://bugs.launchpad.net/ironic/+bug/1526403 https://bugs.launchpad.net/ironic/+bug/1618754

The current Ironic implementation only supports flat networks. For isolation of tenant networks Ironic should be able to pass information to Neutron to allow for provisioning of the baremetal server onto the tenant network.

Problem description

Ironic currently provisions servers only on flat networks hence providing no network isolation between tenants. Ironic should allow end users to utilize a baremetal instance in the same isolated (e.g. VLAN, VXLAN) networks as their virtual machines are.

In order to provide the required network isolation, Ironic should be able to provide the requisite connectivity information (using LLDP) to the Neutron ML2 plugin to allow drivers to provision the top-of-rack (ToR) switch for the baremetal server.

Ironic also poses new challenges for Neutron, for example, the concepts of link aggregation and multiple networks per node. This spec covers the concept of link aggregation where multiple interfaces on the bare metal server connect to switch ports of a single LAG and belong to the same network. However,this spec does not cover:

- the case of trunked ports belonging to multiple networks - this will need a separate spec and implementation in Neutron, see vlan-aware-vms.
- the case where the bare metal server has multiple interfaces that belong to different networks. This may require the capability to specify which NICs belong to which network. It is in scope but the method of deciding which interface gets which network will be decided elsewhere.

Proposed change

When nodes are enrolled/introspected in Ironic, local link (e.g. LLDP) information should be recorded for each port. The option of creating a portgroup to specify an aggregation of said ports should be made available.

Ironic should then send the local link information to Neutron in the form of a port binding profile. For each portgroup, the port binding profile will contain a list of the local link information for each port in that portgroup. For each port not belonging to a portgroup, the port binding profile will contain only the local link information for that particular port. Therefore the length of the list of local link information can be used by Neutron to determine whether to plumb the network for a single port or for a portgroup (in the case of a portgroup there may be additional switch configurations necessary to manage the associated LAG). Each port binding profile sent to the Neutron port create API will result in the creation of a single Neutron port. In this way, each portgroup can represent a LAG for which all member switch ports will belong to the same network and to the same network segment.

The node should first be placed on a provisioning network and then onto the tenant network as specified

in the network-provider spec[1]. To help facilitate this there will be a variable stored in the binding profile to record whether a port binding has been requested. This will allow Ironic to defer binding until Ironic is able to populate the extra information that is required by Neutron.

Note also, there will be no support for PXE booting after deploy (i.e. local disk installation only). The reason for this is because the nodes will not be able to PXE boot if they cannot reach the TFTP server. Local boot will need to be used for any node deployed outside of the provisioning network. Instance netboot should not be permitted when this feature is used. This limitation could be addressed if routing was set up from every tenant network to the TFTP server or if Ironic would work with a TFTP server per tenant network, but these options are out of scope in this spec.

Note

It may be possible for virtual media drivers to support both netboot and localboot when provisioning/tenant networks are isolated. This is because virtual media drivers boot the bare metal using out of band means using BMC and hence bare metal servers NICs dont require to access Ironic conductor when booting the instance.

A supportive ML2 driver can then use the information supplied to provision the port.

The Ironic port object will be updated with a new local_link_connection field dict supplying the following values:

- switch_id
- port_id
- switch_info

These fields are in string format. The information could be gathered by using LLDP based on the LLDP TLV counterparts of chassis_id, port_id and system_name, although it is not strictly required to populate these fields with LLDP values. For example, the switch_id field is used to identify a switch and could be an LLDP based MAC address or an OpenFlow based datapath_id. The switch_info field could be used to distinguish different switch models or some other vendor specific identifier. The switch_id and port_id fields are required and the switch_info field can be optionally populated. The key point is that Ironic and Neutron should share the same understanding of this information in order to connect the Ironic instance to the appropriate Neutron network.

To facilitate link aggregation a new portgroup object will be created. In addition to the base object it will have the following fields:

- id
- uuid
- name
- node_id
- address
- mode
- properties
- extra

[1] http://specs.openstack.org/openstack/ironic-specs/specs/6.1/network-provider.html

- internal_info
- standalone_ports_supported

The `address` field represents the MAC address for bonded NICs of the bare metal server. It is optional, as its value is highly dependent on the instance OS. In case of using ironic through nova, the `address` gets populated by whatever value is generated by neutron for the VIFs address representing this portgroup, if there is one. This happens during every VIF attach call. In case of standalone mode, it can be always `None`, as it should be configured on the instance through the configdrive.

The `mode` field is used to specify the bond mode. If not provided in the portgroup creation request, its default value is determined by the `[DEFAULT]default_portgroup_mode` configuration option, which in turn has a default value of `active-backup`. For a portgroup that was created in an older API version, this configuration value will be used as the value for that portgroups mode. `active-backup` is chosen as the default because this mode does not require any additional configuration on the switch side.

The `properties` field is a dictionary that can contain any parameters needed to configure the portgroup.

For additional information about `mode` and `properties` fields contents, see linux kernel bonding documentation.

The `extra` field can be used to hold any additional information that operators or developers want to store in the portgroup.

The `internal_info` field is used to store internal metadata. This field is read-only.

The `standalone_ports_supported` indicates whether ports that are members of this portgroup can be used as stand-alone ports.

The Ironic port object will then have the following fields added to support new functionality:

- local_link_connection
- portgroup_id
- pxe_enabled

If there are multiple pxe_enabled ports or portgroups, dhcpboot options will be set for all portgroups and all pxe_enabled ports not belonging to any portgroup.

The following port binding related information needs to be passed to Neutron:

Field Name	Description
vnic_type	Type of the profile (baremetal in this case). This would allow at least basic filtering for Ironic ports by ML2 drivers.
local_link_infor	A list of local link connection information either from all Ironic ports in a particular portgroup or from a single Ironic port not belonging to any portgroup.
host_id	This should be set to the Ironic node uuid.

A JSON example to describe the structure is:

```
{
  "port": {
    <all other fields>,
    "vnic_type": "baremetal",
    "host_id": <Ironic node UUID>,
```

(continues on next page)

(continued from previous page)

```
    "binding:profile": {
      "local_link_information": [
        {
          "switch_id": xxx,
          "port_id": xxx,
          "switch_info": zzz,
          <optional more information>
        },
        {
          "switch_id": xxx,
          "port_id": yyy,
          "switch_info": zzz,
          <optional more information>
        }
      ]
      <some more profile fields>
    }
  }
}
```

Alternatives

The current model of prescribing flat networks could be maintained with the same flat network being used for everything. This is not so much an alternative to the proposal in this spec, but rather staying with the existing solution.

Data model impact

The proposed change will be to add the following fields to the port object with their data type and default value for migrations:

Field Name	Field Type	Migration Value
local_link_connection	dict_or_none	None
portgroup_id	int_or_none	None
pxe_enabled	bool	True

All existing ports will have `pxe_enabled` set to `true` so that the current behavior is not changed. The portgroup relationship is a 1:n relationship with the port.

The portgroup object is proposed with the following fields and data types:

Field Name	Field Type
id	int
uuid	str
name	str_or_none
node_id	int_or_none
address	str_or_none
mode	str_or_none
properties	dict_or_none
extra	dict_or_none
internal_info	dict_or_none
standalone_ports_supported	bool
created_at	datetime_or_str_or_none
updated_at	datetime_or_str_or_none

Note

While `mode` attribute of the portgroup object has type str_or_none, its value can not be `None`, unless the database was changed manually. It gets populated either during the database migration, or during portgroup creation (if not specified explicitly). In both cases it is set to the `[DEFAULT]default_portgroup_mode` configuration option value.

State Machine Impact

The state machine will not be directly impacted, however, changes to the new portgroup object and additions of portgroups will only be allowed when a node is in a particular set of states.

Change to port membership of a portgroup can be made when the node is in a MANAGEABLE/INSPECTING/ENROLL state. Any port updates that update local_link_connection or pxe_enabled can only be made when the node is in a MANAGEABLE/INSPECTING/ENROLL state. The reason for limiting to these states is because updating these new port attributes should result in an update of local_link_information in the binding_profile, which would trigger an update in Neutron. It might be safest to only allow this when the node is not in a state where uninterrupted connectivity is expected. These limitations will also ensure that Neutron port updates should only happen during a state change and not automatically with any port-update call.

REST API impact

The following port API methods will be affected:

- `/v1/ports`
  - Retrieve a list of ports.
  - Method type GET.
  - The http response code(s) are unchanged. An additional reason for the 404 error http response code would be if the portgroup resource is specified but is not found.
  - New parameter can be included:
    - `portgroup (uuid_or_name)` - UUID or logical name of a portgroup to only get ports for that portgroup.

  - Body:
    - None
  - Response:
    - JSON schema definition of Port
- `/v1/ports/(port_uuid)`
  - Retrieve information about the given port.
  - Method type GET.
  - The http response code(s) are unchanged.
  - Parameter:
    - `port_uuid (uuid)` - UUID of the port.
  - Body:
    - None
  - Response:
    - JSON schema definition of Port
- `/v1/ports`
  - Create a new port.
  - Method type POST.
  - The http response code(s) are unchanged.
  - Parameter:
    - None
  - Body:
    - JSON schema definition of Port
  - Response:
    - JSON schema definition of Port
- `/v1/ports/(port_uuid)`
  - Update an existing port.
  - Method type PATCH.
  - The http response code(s) are unchanged.
  - Parameter:
    - `port_uuid (uuid)` - UUID of the port.
  - Body:
    - JSON schema definition of PortPatch
  - Response:
    - JSON schema definition of Port

- JSON schema definition of Port (data sample):

```
{
  "address": "fe:54:00:77:07:d9",
  "created_at": "2015-05-12T10:00:00.529243+00:00",
  "extra": {
    "foo": "bar",
  },
  "links": [
    {
      "href": "http://localhost:6385/v1/ports/
       1004e542-2f9f-4d9b-b8b9-5b719fa6613f",
      "rel": "self"
    },
    {
      "href": "http://localhost:6385/ports/
       1004e542-2f9f-4d9b-b8b9-5b719fa6613f",
      "rel": "bookmark"
    }
  ],
  "node_uuid": "e7a6f1e2-7176-4fe8-b8e9-ed71c77d74dd",
  "updated_at": "2015-05-15T09:04:12.011844+00:00",
  "uuid": "1004e542-2f9f-4d9b-b8b9-5b719fa6613f",
  "local_link_connection": {
    "swwitch_id": "0a:1b:2c:3d:4e:5f",
    "port_id": "Ethernet3/1",
    "switch_info": "switch1",
  },
  "portgroup_uuid": "6eb02b44-18a3-4659-8c0b-8d2802581ae4",
  "pxe_enabled": true
}
```

- JSON schema definition of PortPatch would be a subset of JSON schema of Port.

The following API methods will be added in support of the new portgroup model:

- `/v1/portgroups`
  - Retrieve a list of portgroups.
  - Method type GET.
  - Normal http response code will be 200.
  - Expected error http response code(s):
    - 400 for bad query or malformed syntax (e.g. if address is not mac-address format)
    - 404 for resource (e.g. node) not found
  - Parameters:
    - `node (uuid_or_name)` - UUID or name of a node, to only get portgroups for that node.
    - `address (macaddress)` - MAC address of a portgroup, to only get portgroup which has this MAC address.

        * `marker (uuid)` - pagination marker for large data sets.
        * `limit (int)` - maximum number of resources to return in a single result.
        * `sort_key (unicode)` - column to sort results by. Default: id.
        * `sort_dir (unicode)` - direction to sort. asc or desc. Default: asc.
    - Body:
        * None
    - Response:
        * JSON schema definition of PortgroupCollection
- `/v1/portgroups/(portgroup_ident)`
    - Retrieve information about the given portgroup.
    - Method type GET.
    - Normal http response code will be 200.
    - Expected error http response code(s):
        * 400 for bad query or malformed syntax
        * 404 for resource (e.g. portgroup) not found
    - Parameters:
        * `portgroup_ident (uuid_or_name)` - UUID or logical name of a portgroup.
    - Body:
        * None
    - Response:
        * JSON schema definition of Portgroup
- `/v1/portgroups`
    - Create a new portgroup.
    - Method type POST.
    - Normal http response code will be 201.
    - Expected error http response code(s):
        * 400 for bad query or malformed syntax
        * 409 for resource conflict (e.g. if portgroup name already exists because the name should be unique)
    - Parameters:
        * None
    - Body:
        * JSON schema definition of Portgroup
    - Response:
        * JSON schema definition of Portgroup

- `/v1/portgroups/(portgroup_ident)`
  - Delete a portgroup.
  - Method type DELETE.
  - Normal http response code will be 204.
  - Expected error http response code(s):
    - 400 for bad query or malformed syntax
    - 404 for resource (e.g. portgroup) not found
  - Parameters:
    - `portgroup_ident (uuid_or_name)` - UUID or logical name of a portgroup.
  - Body:
    - None
  - Response:
    - N/A
- `/v1/portgroups/(portgroup_ident)`
  - Update an existing portgroup.
  - Method type PATCH.
  - Normal http response code will be 200.
  - Expected error http response code(s):
    - 400 for bad query or malformed syntax
    - 404 for resource (e.g. portgroup) not found
    - 409 for resource conflict (e.g. if portgroup name already exists because the name should be unique)
  - Parameters:
    - `portgroup_ident (uuid_or_name)` - UUID or logical name of a portgroup.
  - Body:
    - JSON schema definition of PortgroupPatch
  - Response:
    - JSON schema definition of Portgroup
- `/v1/portgroups/detail`
  - Retrieve a list of portgroups with detail. The additional detail option would return all fields, whereas without it only a subset of fields would be returned, namely uuid and address.
  - Method type GET.
  - Normal http response code will be 200.
  - Expected error http response code(s):
    - 400 for bad query or malformed syntax

        * 404 for resource (e.g. node) not found
    - Parameters:
        * `node (uuid_or_name)` - UUID or name of a node, to only get portgroups for that node.
        * `address (macaddress)` - MAC address of a portgroup, to only get portgroup which has this MAC address.
        * `marker (uuid)` - pagination marker for large data sets.
        * `limit (int)` - maximum number of resources to return in a single result.
        * `sort_key (unicode)` - column to sort results by. Default: id.
        * `sort_dir (unicode)` - direction to sort. asc or desc. Default: asc.
    - Body:
        * None
    - Response:
        * JSON schema definition of PortgroupCollection with detail.
- `/v1/nodes/(node_ident)/portgroups`
    - Retrieve a list of portgroups for node.
    - Method type GET.
    - Normal http response code will be 200.
    - Expected error http response code(s):
        * 400 for bad query or malformed syntax
        * 404 for resource (e.g. node) not found
    - Parameters:
        * `node_ident (uuid_or_name)` - UUID or logical name of a node.
    - Body:
        * None
    - Response:
        * JSON schema definition of PortgroupCollection.
- `/v1/nodes/(node_ident)/portgroups/detail`
    - Retrieve a list of portgroups with detail for node.
    - Method type GET.
    - Normal http response code will be 200.
    - Expected error http response code(s):
        * 400 for bad query or malformed syntax
        * 404 for resource (e.g. node) not found
    - Parameters:
        * `node_ident (uuid_or_name)` - UUID or logical name of a node.

  - Body:
    - None
  - Response:
    - JSON schema definition of PortgroupCollection with detail.

- `/v1/portgroups/(portgroup_ident)/ports`
  - Retrieve a list of ports for portgroup.
  - Method type GET.
  - Normal http response code will be 200.
  - Expected error http response code(s):
    - 400 for bad query or malformed syntax
    - 404 for resource (e.g. portgroup) not found
  - Parameters:
    - `portgroup_ident (uuid_or_name)` - UUID or logical name of a portgroup.
  - Body:
    - None
  - Response:
    - JSON schema definition of PortCollection.

- `/v1/portgroups/(portgroup_ident)/ports/detail`
  - Retrieve a list of ports with detail for portgroup.
  - Method type GET.
  - Normal http response code will be 200.
  - Expected error http response code(s):
    - 400 for bad query or malformed syntax
    - 404 for resource (e.g. portgroup) not found
  - Parameters:
    - `portgroup_ident (uuid_or_name)` - UUID or logical name of a portgroup.
  - Body:
    - None
  - Response:
    - JSON schema definition of PortCollection with detail.

- JSON schema definition of Portgroup (data sample):

```
{
  "address": "fe:54:00:77:07:d9",
  "created_at": "2015-05-12T10:10:00.529243+00:00",
  "extra": {
```

(continues on next page)

(continued from previous page)


```
    "foo": "bar",
  },
  "internal_info": {},
  "links": [
    {
      "href": "http://localhost:6385/v1/portgroups/
      6eb02b44-18a3-4659-8c0b-8d2802581ae4",
      "rel": "self"
    },
    {
      "href": "http://localhost:6385/portgroups/
      6eb02b44-18a3-4659-8c0b-8d2802581ae4",
      "rel": "bookmark"
    }
  ],
  "mode": "802.3ad",
  "name": "node1_portgroup1",
  "node_uuid": "e7a6f1e2-7176-4fe8-b8e9-ed71c77d74dd",
  "ports": [
    {
      "href": "http://127.0.0.1:6385/v1/portgroups/
      6eb02b44-18a3-4659-8c0b-8d2802581ae4/ports",
      "rel": "self"
    },
    {
      "href": "http://127.0.0.1:6385/portgroups/
      6eb02b44-18a3-4659-8c0b-8d2802581ae4/ports",
      "rel": "bookmark"
    }
  ],
  "properties": {
    "bond_xmit_hash_policy": "layer3+4",
    "bond_miimon": 100
  },
  "standalone_ports_supported": true,
  "updated_at": "2015-05-15T09:04:12.011844+00:00",
  "uuid": "6eb02b44-18a3-4659-8c0b-8d2802581ae4"
}
```


- JSON schema definition of PortgroupCollection:


```
{
  "portgroups": [
      {
          "address": "fe:54:00:77:07:d9",
          "links": [
              {
                  "href": "http://localhost:6385/v1/portgroups/
                  6eb02b44-18a3-4659-8c0b-8d2802581ae4",
                  "rel": "self"
```


(continues on next page)

(continued from previous page)

```
            },
            {
                "href": "http://localhost:6385/portgroups/
                6eb02b44-18a3-4659-8c0b-8d2802581ae4",
                "rel": "bookmark"
            }
        ],
        "name": "node1_portgroup1",
        "uuid": "6eb02b44-18a3-4659-8c0b-8d2802581ae4"
    }
  ]
}
```

- JSON schema definition of PortgroupCollection with detail:

```
{
  "portgroups": [
    {
      "address": "fe:54:00:77:07:d9",
      "created_at": "2016-08-18T22:28:48.165105+00:00",
      "extra": {},
      "internal_info": {},
      "links": [
        {
          "href": "http://127.0.0.1:6385/v1/portgroups/
          6eb02b44-18a3-4659-8c0b-8d2802581ae4",
          "rel": "self"
        },
        {
          "href": "http://127.0.0.1:6385/portgroups/
          6eb02b44-18a3-4659-8c0b-8d2802581ae4",
          "rel": "bookmark"
        }
      ],
      "mode": "802.3ad",
      "name": "node1_portgroup1",
      "node_uuid": "e7a6f1e2-7176-4fe8-b8e9-ed71c77d74dd",
      "ports": [
        {
          "href": "http://127.0.0.1:6385/v1/portgroups/
          6eb02b44-18a3-4659-8c0b-8d2802581ae4/ports",
          "rel": "self"
        },
        {
          "href": "http://127.0.0.1:6385/portgroups/
          6eb02b44-18a3-4659-8c0b-8d2802581ae4/ports",
          "rel": "bookmark"
        }
      ],
      "properties": {
```

(continues on next page)

(continued from previous page)

```
        "bond_xmit_hash_policy": "layer3+4",
        "bond_miimon": 100
      },
      "standalone_ports_supported": true,
      "updated_at": "2016-11-04T17:46:09+00:00",
      "uuid": "6eb02b44-18a3-4659-8c0b-8d2802581ae4"
    }
  ]
}
```

- JSON schema definition of PortgroupPatch would be a subset of JSON schema of Portgroup.

Does the API microversion need to increment?

- Yes.

Example use case including typical API samples for both data supplied by the caller and the response.

- Example of port create.
  - Data supplied:

```
{
  "address": "fe:54:00:77:07:d9",
  "node_uuid": "e7a6f1e2-7176-4fe8-b8e9-ed71c77d74dd",
  "local_link_connection": {
    "switch_id": "0a:1b:2c:3d:4e:5f",
    "port_id": "Ethernet3/1",
    "switch_info": "switch1",
    },
  "pxe_enabled": true
}
```

  - Response 201 with body:

```
{
  "address": "fe:54:00:77:07:d9",
  "node_uuid": "e7a6f1e2-7176-4fe8-b8e9-ed71c77d74dd",
  "local_link_connection": {
    "switch_id": "0a:1b:2c:3d:4e:5f",
    "port_id": "Ethernet3/1",
    "switch_info": "switch1",
    },
  "pxe_enabled": true
  "created_at": "2015-05-12T10:00:00.529243+00:00",
  "extra": {
  },
  "links": [
    {
      "href": "http://localhost:6385/v1/ports/
       1004e542-2f9f-4d9b-b8b9-5b719fa6613f",
      "rel": "self"
```

(continues on next page)

(continued from previous page)


```
    },
    {
      "href": "http://localhost:6385/ports/
       1004e542-2f9f-4d9b-b8b9-5b719fa6613f",
      "rel": "bookmark"
    }
  ],
  "updated_at": null,
  "uuid": "1004e542-2f9f-4d9b-b8b9-5b719fa6613f",
  "portgroup_uuid": null,
}
```


- Example of portgroup create.
  - Data supplied:


```
{
  "address": "fe:54:00:77:07:d9",
  "node_uuid": "e7a6f1e2-7176-4fe8-b8e9-ed71c77d74dd",
  "standalone_ports_supported": true,
  "name": "node1_portgroup1"
}
```


  - Response 201 with body:


```
{
  "address": "fe:54:00:77:07:d9",
  "node_uuid": "e7a6f1e2-7176-4fe8-b8e9-ed71c77d74dd",
  "name": "node1_portgroup1"
  "created_at": "2015-05-12T10:10:00.529243+00:00",
  "extra": {
  },
  "internal_info": {},
  "links": [
    {
      "href": "http://localhost:6385/v1/portgroups/
       6eb02b44-18a3-4659-8c0b-8d2802581ae4",
      "rel": "self"
    },
    {
      "href": "http://localhost:6385/portgroups/
       6eb02b44-18a3-4659-8c0b-8d2802581ae4",
      "rel": "bookmark"
    }
  ],
  "mode": null,
  "ports": [
    {
      "href": "http://127.0.0.1:6385/v1/portgroups/
      6eb02b44-18a3-4659-8c0b-8d2802581ae4/ports",
```


(continues on next page)

(continued from previous page)

```
      "rel": "self"
    },
    {
      "href": "http://127.0.0.1:6385/portgroups/
      6eb02b44-18a3-4659-8c0b-8d2802581ae4/ports",
      "rel": "bookmark"
    }
  ],
  "properties": {},
  "standalone_ports_supported": true,
  "updated_at": null,
  "uuid": "6eb02b44-18a3-4659-8c0b-8d2802581ae4",
}
```

- Example of port update.
  - Parameter port_uuid=1004e542-2f9f-4d9b-b8b9-5b719fa6613f
  - Data supplied (JSON PATCH syntax where op can be add/replace/delete):

```
[{"path": "/portgroup_uuid", "value":
  "6eb02b44-18a3-4659-8c0b-8d2802581ae4", "op": "add"}]
```

  - Response 200 with body:

```
{
  "address": "fe:54:00:77:07:d9",
  "node_uuid": "e7a6f1e2-7176-4fe8-b8e9-ed71c77d74dd",
  "local_link_connection": {
    "switch_id": "0a:1b:2c:3d:4e:5f",
    "port_id": "Ethernet3/1",
    "switch_info": "switch1",
    },
  "pxe_enabled": true
  "created_at": "2015-05-12T10:00:00.529243+00:00",
  "extra": {
  },
  "links": [
    {
      "href": "http://localhost:6385/v1/ports/
       1004e542-2f9f-4d9b-b8b9-5b719fa6613f",
      "rel": "self"
    },
    {
      "href": "http://localhost:6385/ports/
       1004e542-2f9f-4d9b-b8b9-5b719fa6613f",
      "rel": "bookmark"
    }
  ],
  "updated_at": "2015-05-12T10:20:00.529243+00:00",
  "uuid": "1004e542-2f9f-4d9b-b8b9-5b719fa6613f",
```

(continues on next page)

(continued from previous page)

```
  "portgroup_uuid": "6eb02b44-18a3-4659-8c0b-8d2802581ae4",
}
```

  - Note that the port update API should support updating the portgroup_id of the port object. This will allow operators to migrate existing deployments.

- Example of port list.
  - Parameter node_uuid=e7a6f1e2-7176-4fe8-b8e9-ed71c77d74dd
  - Response 200 with body:

```
{"ports": [
  {
  "address": "fe:54:00:77:07:d9",
  "links": [
    {
      "href": "http://localhost:6385/v1/ports/
       1004e542-2f9f-4d9b-b8b9-5b719fa6613f",
      "rel": "self"
    },
    {
      "href": "http://localhost:6385/ports/
       1004e542-2f9f-4d9b-b8b9-5b719fa6613f",
      "rel": "bookmark"
    }
  ],
  "uuid": "1004e542-2f9f-4d9b-b8b9-5b719fa6613f",
  "portgroup_uuid": "6eb02b44-18a3-4659-8c0b-8d2802581ae4",
  }
]}
```

  - Note that portgroup_uuid is now returned in the response.

Discuss any policy changes, and discuss what things a deployer needs to think about when defining their policy.

- Ironic has an admin-only policy so policy definitions should not be a concern.
- A deployer should be aware of the capabilities of the particular ML2 driver for supporting use of the new local_link_information that will be passed to it via the binding_profile.

Is a corresponding change in the client library and CLI necessary?

- The client library and CLI should be updated to support the new APIs.

Is this change discoverable by clients? Not all clients will upgrade at the same time, so this change must work with older clients without breaking them.

- The changes to the API will be backward-compatible so older clients will still continue to work as-is.

Client (CLI) impact

The python-ironicclient and OSC would need updated to support the new portgroups APIs.

In the commands below, <portgroup> means that this placeholder can contain both portgroup UUID or name. <portgroup_uuid> can contain only portgroup UUID.

Example usage of the new methods:

- For ports, the CLI would support port creation with new optional parameters specifying the new port attributes (local_link_connection, portgroup_id and pxe_enabled) and would also support update of these attributes. As examples:

  ironic CLI:

  - ironic port-create -a <address> -n <node> [-e <key=value>] [local-link-connection <key=value>] [portgroup <portgroup>] [pxe-enabled <boolean>]
  - ironic port-update <port_uuid> add portgroup_uuid=<portgroup_uuid> local-link-connection <key=value> pxe-enabled <boolean>

  openstack baremetal CLI:

  - openstack baremetal port create node <node> [local-link-connection <key=value>] [port-group <portgroup>] [pxe-enabled <boolean>] <address>
  - openstack baremetal port set [port-group <portgroup>] [local-link-connection <key=value>] [pxe-enabled <boolean>] <port>
  - openstack baremetal port list [address <mac-address>] [node <node> | port-group <port-group>]

- For portgroups, the CLI would support the following new methods:

  ironic CLI:

  - ironic portgroup-create node <node> [address <mac-address>] [name <portgroupname>] [-e <key=value>] [standalone-ports-supported <boolean>] [-m <mode>] [-p <key=value>]
  - ironic portgroup-delete <portgroup> [<portgroup>]
  - ironic portgroup-list [detail | fields <field> [<field>]] [node <node>] [address <mac-address>] [limit <limit>] [marker <portgroup_uuid>] [sort-key <field>] [sort-dir <direction>]
  - ironic portgroup-port-list [detail | fields <field> [<field>]] [limit <limit>] [marker <port-group_uuid>] [sort-key <field>] [sort-dir <direction>] <portgroup>
  - ironic portgroup-show [address] [fields <field> [<field>]] <id>
    - <id> is the UUID or name of the portgroup (or MAC address if address is specified)
  - ironic portgroup-update <portgroup> <op> <path=value> [<path=value>]
    - <op> is add, remove or replace.
    - <path=value> is the attribute to add, remove or replace. Can be specified multiple times. For remove only <path> is necessary.
  - Note: Even though the ironic CLI includes ironic node-port-list, we are NOT going to provide a corresponding ironic node-portgroup-list. Rather, the list of portgroups of a node will be available via ironic portgroup-list node.

openstack baremetal CLI:

- openstack baremetal port group create node <uuid> [name <name>] [extra <key=value>] [support-standalone-ports | unsupport-standalone-ports] [mode <mode>] [properties <key=value>] [address <mac-address>]
- openstack baremetal port group delete <portgroup> [<portgroup>]
- openstack baremetal port group list [marker <portgroup>] [address <mac-address>] [node <node>] [sort <key>[:<direction>]] [long | fields <field> [<field>]]
- openstack baremetal port group show [address] [fields <field> [<field>]] <id>
  - <id> is the UUID or name of the portgroup (or MAC address if address is specified)
- openstack baremetal port group set [address <mac-address>] [name <name>] [node <node>] [extra <key=value>] [support-standalone-ports | unsupport-standalone-ports] [mode <mode>] [properties <key=value>] <portgroup>
- openstack baremetal port group unset [address] [name] [extra <key>] [properties key] <portgroup>
- To add ports to a portgroup, the portgroup should first be created and then port_update or port create called.

The python-ironicclient would also need the Port detailed resource extended to include the new port attributes.

RPC API impact

No impact on existing API calls.

New RPC API calls would be needed:

- update_portgroup
- destroy_portgroup

These new API calls will use call(). As for the existing API call for update_port, the new API call for update_portgroup should request an update for DHCP if the address field is updated.

To roll this change out to an existing deployment, the ironic-conductor should be upgraded before the ironic-api.

Driver API impact

The NeutronDHCPApi class in `ironic/dhcp/neutron` updates Neutron ports with DHCP options. The vifs are obtained in `ironic/common/network` by extracting `vif_port_id` from the `extra` attributes of Ironic ports. This method should be updated if vifs are bound to portgroups as well as ports.

The complementary network-provider spec[Page 1195, 1] provides details regarding the workflow of the network flip and the point at which the binding profile will be passed to Neutron to bind the port.

Nova driver impact

As this work depends on the attach/detach interface work[2], the only thing that needs to be changed to fully support portgroups is configdrive generation.

[2] http://specs.openstack.org/openstack/ironic-specs/specs/approved/interface-attach-detach-api.html

Nova will call into ironic to get the list of ports of each portgroup that has a VIF associated with it, along with portgroup `mode` and `properties` fields (see *Data model impact* section), and update the network metadata with the needed information. When the contents of the `properties` dictionary gets passed to the config drive builder in nova, we will ensure that `bond_` prefix is prepended to all key names, so that these keys are not ignored by cloud-init when reading the config drive.

Ramdisk impact

N/A

Security impact

The new REST API calls for portgroups should not be usable by the end user. Only operators and administrators should be able to manage portgroups and local_link_connection data of ports, because these settings are used to configure the network. This is satisfied because Ironic is an admin-only API, so there should be no security impact.

Other end user impact

Using the binding profile to enable flipping between provisioning and tenant networks means there will be no support for PXE booting after deploy (i.e. local disk installation only). How to allow operators to deploy instances using either net-boot or local boot using the same Ironic conductor should be discussed in the complementary network-provider spec[Page 1195, 1].

Scalability impact

There will be more API calls made to Ironic in order to create and use portgroups but impact on scalability should be negligible.

Performance Impact

None.

Other deployer impact

New database columns are added to the port table and a new database table portgroup is introduced, so this will require a database migration.

Deployers will need to deploy an ML2 mechanism driver that supports connecting baremetal resources to Neutron networks.

If using Nova, deployers will need to deploy a version of Nova that supports this feature.

Deployers may want to set the `[DEFAULT]default_portgroup_mode` configuration option to match their environment. Its default value is `active-backup`.

Deployers should be aware that automated upgrade or migration for already-provisioned nodes is not supported. Deployers should follow this recommendation for upgrading a node in an existing deployment to use this new feature:

- Upgrade the OpenStack services.
- Move node into the MANAGEABLE state.
- Update node driver field (see the network-provider spec[Page 1195, 1]).

- Create Ironic portgroups.
- Update Ironic port membership to portgroups.
- Update Ironic ports with local_link_connection data.
- Move node into the AVAILABLE state.

Developer impact

Neutron ML2 mechanism drivers should support this feature by using the data passed in binding profile to dynamically configure relevant ports and port-channels on the relevant switch(es).

Implementation

Assignee(s)

- laura-moore
- yhvh (Will Stevenson)
- bertiefulton
- sukhdev-8
- vsaienko
- vdrok

Work Items

- Extend port table.
- Create the new portgroup table.
- Implement extension to port APIs.
- Implement the new portgroup APIs.
- Implement the extension to the RPC API.
- Implement the changes to the Nova driver to get and use the binding profile.
- Implement the changes needed to get vifs for updating Neutron port DHCP options.
- Implement tests for the new functionality.
- Implement updates to the python-ironicclient.
- Update documentation.

Dependencies

Network flip is dependent on the network-provider spec[Page 1195, 1].

Nova changes are dependent on attach/detach interfaces work[Page 1212, 2].

VLAN provisioning on switch(es) is dependent on ML2 driver functionality being developed to support this feature.

Testing

Existing default behaviour will be tested in the gate by default.

New tests will need to be written to test the new APIs and database updates.

Simulation of connecting real hardware to real switches for testing purposes is described in the network-provider spec[Page 1195, 1].

Upgrades and Backwards Compatibility

Default behavior is the current behavior, so this change should be fully backwards compatible.

Documentation Impact

This feature will be fully documented.

References

Discussions on the topic include:

- https://etherpad.openstack.org/p/YVR-neutron-ironic
- https://etherpad.openstack.org/p/liberty-ironic-network-isolation
- https://etherpad.openstack.org/p/network-interface-vifs-configdrive
- Logs from https://wiki.openstack.org/wiki/Meetings/Ironic-neutron

5.15.156 Redfish hardware type and interfaces

https://bugs.launchpad.net/ironic/+bug/1526477

This specification proposes the addition of a new `redfish` hardware type, power and management interfaces in order to support ironic deployment on Redfish compliant servers.

Problem description

The Distributed Management Task Force (DMTF) has published a new specification called Redfish (refer to http://www.dmtf.org/standards/redfish) to provide a RESTful based API to manage servers in a standard way. This specification aims at adding support to ironic for controlling Redfish compliant servers.

Proposed change

This spec proposes adding a new `redfish` hardware type, power and management interfaces. None of which will be enabled by default.

The new interfaces will use the sushy library in order to handle the communication between the driver and the Redfish controller.

This library may be switched to python-redfish in the future after re-evaluation by the ironic community. The switch to python-redfish is outside the scope of this specification, but it should not cause any public interface to be changed. It would most likely involve code changes and new configuration options.

Note that no OEM specific extension will be supported.

Alternatives

None

Data model impact

None

RPC API impact

None

State Machine Impact

None

REST API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

None

Security impact

None

Client (CLI) impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

The following driver_info fields are required while enrolling nodes into ironic:

- `redfish_address`: The URL address to the Redfish controller. It should include scheme and authority portion of the URL. For example: https://mgmt.vendor.com
- `redfish_system_id`: The canonical path to the ComputerSystem resource that the driver will interact with. It should include the root service, version and the unique resource path to a ComputerSystem within the same authority as the redfish_address property. For example: /redfish/v1/Systems/1
- `redfish_username`: User account with admin/server-profile access privilege. Although this property is not mandatory its highly recommended to set a username. Optional
- `redfish_password`: User account password. Although this property is not mandatory its highly recommended to set a password. Optional
- `redfish_verify_ca`: This property contains either a boolean value, a path to a CA_BUNDLE file or directory with certificates of trusted CAs. If set to True the driver will verify the host certificates; if False the driver will ignore verifying the SSL certificate; If its a path the driver will use the specified certificate or one of the certificates in the directory. Defaults to True. Optional

For more information about the expected syntax of the `redfish_system_id` property check the Resource identifier property section of the DMTF specification.

The following new configuration variables are proposed (and their default values) to be added to the conductor variable group:

- [redfish]/`connection_attempts`

  Maximum number of attempts to try to connect to Redfish.

- [redfish]/`connection_retry_interval`

  Number of seconds to wait between attempts to connect to Redfish.

Developer impact

None

Implementation

Assignee(s)

Primary assignee:

- bcornec
- lucasagomes

Other contributors:

- ribaudr

Work Items

- Add a new `redfish` hardware type, power and management interfaces.
- Write unit-tests for the new code.
- Modify the ironic DevStack module to setup a virtual environment that is able to test nodes using the new Redfish driver.
- Write documentation.

Dependencies

The new `redfish` power and management interfaces will require the sushy library to be installed on the conductor node.

Testing

Unit-tests will be implemented for Redfish support.

DevStack will be updated to setup the nodes with the redfish driver and the libvirt mockup that is shipped with Sushy allowing it to be tests in gate against virtual machines.

Upgrades and Backwards Compatibility

This driver will not break any compatibility with either the REST API or the RPC API.

Documentation Impact

- Updating ironic documentation section Enabling Drivers with Redfish related instructions.
- Updating ironic install-guide documentation section Setup the drivers for the Bare Metal service.

References

- Redfish DMTF: http://www.dmtf.org/standards/redfish
- Sushy library: https://github.com/openstack/sushy
- python-redfish library: https://github.com/openstack/python-redfish

5.15.157 Redfish Proxy for Ironic Node Power Controls

https://storyboard.openstack.org/#!/story/2009184

One important aspect of our goal of improving support for node multi-tenancy is to provide a seamless user experience for node lessees. This includes enabling lessees to use their existing provisioning workflows on leased nodes and to have these tools work as expected. Ironic policy does allow for lessees to access basic power controls through either the Ironic REST API or the `openstack baremetal` CLI; however, if such tools lack instructions for communicating with the Ironic API, the only way for them to function is by making requests directly to the BMC of the node they wish to provision. This would require giving the lessee BMC power credentials, which is especially problematic considering that one of the goals of node multi-tenancy is to enable this sort of functionality without giving the user such low-level access. [NMTSTORY] [NODELEAS]

Because such a solution is undesirable, to enable the use of existing provisioning tools, we instead intend to emulate a standards-based interface (Redfish, in this case) for use by node lessees. This interface will receive Redfish API calls such as those made by provisioning tools, perform the corresponding action

within Ironic, and return a response in the format defined by the Redfish schema. Redfish clients will be able to authenticate in the way they expect to and these requests will be handled using the same authentication strategy as the rest of the existing Ironic infrastructure. If all goes according to plan, this feature will solve the compatibility problem previously described, and will do so without giving any special permissions to the lessee.

Problem description

The main concern here is compatibility the problem in which a lessee has tools that expect to communicate via an inaccessible interface has two undesirable solutions.

The first is for the lessees workflow to be refactored to instead make use of the Ironic API. For the user this is both time and resource-consuming, especially considering these lessees will often have access to bare metal nodes for a limited timeframe.

The second is to provide the lessee with BMC credentials, which aside from going against the principles of node multi-tenancy, is a huge security risk. If these credentials are not limited in access scope, lessees could possibly damage or otherwise compromise the bare metal node. Additionally, if the provided credentials are not immediately revoked when the lessee is supposed to lose access to the node, they would retain access to the nodes power controls (and potentially more) until the credentials are changed.

These solutions potentially address the compatibility issue, but in turn, they create either a major efficiency issue or a major security issue. This feature intends to address compatibility without significantly compromising security or efficiency.

Proposed change

We will create a new WSGI service within the main Ironic repo which will serve a REST API at `/redfish` that to the end user, will function like a legitimate v1 Redfish endpoint, albeit with a limited feature set. This interface will provide minimal functionality, just enough to allow Redfish provisioning tools to set the power state of an Ironic node. In the future, this service could be extended to provide additional features, such as boot devices, boot modes, and virtual media; however, this is beyond the scope of what we aim to achieve in this spec.

Implementation details

The three key components necessary to achieve this goal are:

1. a means by which users can be authenticated
2. a way to determine the users identity (roles, projects, privileges, etc.)
3. interfaces for performing operations on the nodes the user wishes to provision.

Broadly, the workflow for making use of this feature shall be as follows:

1. Acquire credentials to be used for authenticating with the Ironic Redfish intermediary.
    - If the Ironic system in question uses Keystone for authentication, the user must create a set of application credentials scoped to the appropriate project(s) and domain.
    - If not, the user will authenticate via HTTP basic authentication, which will be handled using Ironics basic auth middleware. Configuring the backend where credentials are stored will be left up to individual Ironic system operators.
2. (Keystone users only) Authenticate using the Redfish SessionService interface.

- Keystone users will authenticate through the Redfish SessionService interface using the UUID of their newly created application credential as a username and the credentials secret as a password. In response, they will receive a Redfish schema-compliant Session object in the body, as well as an authorization token and the URL of the newly created Sessions interface in the header.

3. Perform the necessary provisioning operations.
   - All requests to Systems endpoints will require authentication; users who authenticated with the SessionService in step 2 must provide the `X-Auth-Token` they received in each request header, and users working with HTTP Basic Auth must provide base64-encoded credentials in each request header (as specified in [RFC7617]).
4. (Keystone users only) End the Session created in step 2.
   - Keystone users will send a DELETE request to the URL of the Session object returned to them previously, internally revoking the created Keystone authentication token (note: not the application credential). If the user wishes to perform further actions, they will need to repeat the authentication process from step 2 again.

Authentication

The actions that this feature shall provide access to should only be accessed by certain users. The question of authentication is one of is the person that is requesting access really who they claim to be? How we are to answer this question depends on the authenticaton strategy in place on the Ironic system in question.

In the context of Keystone (v3), the question of who is this person? requires a few pieces of information the users identifier, the identifier of the project theyre working on, and the identifier of the domain in which the user and project exist. However, Redfish expects only the identifier of the user (UUID or username) to determine identity and as such, authentication would end up requiring the Redfish user to provide more information than they would otherwise expect to provide, which poses a potential problem.

We intend to solve this problem through the use of application credentials, which specify the user, project, and domain they are scoped to. Since each application credential possesses a UUID, we can use this identifier in place of all the information that would otherwise be required by Keystone. [APPCREDS] This approach is also beneficial from a security standpoint, as it reduces the number of times raw user credentials are handled directly.

The user will pass the credentials UUID and secret to the SessionService, where it will internally be passed along to Keystone for verification. If the information provided is valid, an authorization token will be created and sent back to the user in a format emulating that of a Redfish Session. Since Redfish Sessions are required to have a UUID, we will use the audit ID of the newly created Keystone auth token to satisfy this requirement. According to the Keystone API reference, You can use these audit IDs to track the use of a token without exposing the token ID to non-privileged users. [KSTNEAPI] We want to make sure this proxy does not unintentionally expose sensitive information, and the decision to use audit IDs seems like a sensible one in the context of this problem.

Once the user is finished performing the provisioning actions they intend to carry out, they will send a DELETE request to the Session URL, as per the Redfish spec. Internally, this will revoke the Keystone authorization token, essentially logging the user out. This is the intended method for ending a user session, however it is important to note the difference between how Keystone and Redfish handle session expiration.

Redfish Sessions are designed to expire after a period of inactivity, while Keystone authorization tokens are designed to expire at a specific time (e.g. an hour or two after creation). We do not intend to mimic

Redfish Session expiration, since we feel the added overhead and code complexity is not worth the minimal benefit this detail would provide. Auth tokens are ephemeral in nature, and it is up to the user to recognize this and account for the case of unexpected expiration, whether we implement this detail or not.

The authentication process for users of HTTP Basic Auth will be simple, as this strategy is standards-based (see [RFC7617]). The user will provide base64-encoded credentials with every request to a Redfish endpoint that expects a user to be authorized. Since Ironic supports basic authentication, implementing this will simply be a matter of passing the users credentials through the pre-existing basic auth middleware. Additionally, if basic auth is in use, the SessionService will be disabled and unusable.

Identity

Since application credentials are scoped upon creation, obtaining the pieces of information that constitute a users identity should be a straightforward process using the existing Ironic policy code and the Keystone middleware. We will use this information to determine what data, actions, etc. the user has access to via the same rules and methods as the existing Ironic API.

It is important to note that with basic auth, such policy-based access restrictions are essentially non-existent. If a user can log in, they will have access to all available data. However, since our basic auth strategy *is* Ironics basic auth, any extension to Ironics basic auth capability would in turn be an extension to the capability of this feature.

Provisioning Tools

The node provisioning tools that will be implemented here shall be functionally identical to existing Bare Metal endpoints, as shown here. The internal logic for achieving this functionality shall mirror that of the actual Ironic API as closely as possible; in theory the only difference should be in how requests by the user and responses to the user are formatted.

Emulated Redfish URI	Equivalent Ironic URI
[GET] /redfish/v1/SystemService/Systems	[GET] /v1/nodes
[GET] /redfish/v1/SystemService/Systems/{uuid}	[GET] /v1/nodes/{uuid}
[POST] /redfish/v1/SystemService/Systems/{uuid} /Actions/ComputerSystem.Reset	**[PUT] /v1/nodes/{uuid}** /states/power

This intermediary will abide by version 1.0.0 of the Redfish spec [RFSHSPEC] and schema [RFSHSCHM] for maximum backwards compatibility with existing tools. More details regarding the planned functionality of these endpoints will be elaborated upon below in the *REST API Impact* section.

Alternatives

The type of BMC interface emulation were looking to implement here does already exist in sushy-tools [SUSHY] and VirtualBMC [VIRTBMC], which emulate Redfish and IPMI respectively. A previous spec was submitted by Tzu-Mainn Chen (tzumainn) which proposed the idea of a sushy-tools driver in Ironic to enable this functionality, but concerns about security, along with the potential value of this existing in Ironic proper have led to the proposal of this spec. [PREVSPEC]

We currently plan on implementing this as a separate WSGI service within the Ironic repository, however

it is possible to have both the Ironic API and this Redfish proxy run under the same service. Since both are separate, independent WSGI apps, a WSGI dispatcher, such as the Werkzeug application dispatcher middleware [WSGIDISP] could be used to achieve this.

Data model impact

None.

State Machine Impact

None.

REST API impact

No changes will be made to the Ironic API proper, rather, a new WSGI service hosting a new API will be created as described below. End-users shall be able to interact with this API as if it were a v1.0.0 Redfish endpoint (see [RFSHSPEC] and [RFSHSCHM]).

Since this is a new service, Ironic operators will need to account for the fact that it will need its own port and (if using Keystone) will need to be added as a new endpoint within Keystone. If this proves to be a significant enough inconvenience, however, it could be possible to launch both the Ironic API and this proxy within one service as described above under *Alternatives*.

Redfish API Versions:

- GET /redfish
  - Returns the Redfish protocol version (v1). This will always return the same response shown below, as per the Redfish API spec. (section 6.2 of [RFSHSPEC])
  - Normal response code: 200 OK
  - Example response:

```
{
    "v1": "/redfish/v1/"
}
```

Name	Type	Description
v1	string	The URL of the Redfish v1 ServiceRoot.

- GET /redfish/v1/
  - The Redfish service root URL, will return a Redfish ServiceRoot object containing information about what is available on the Redfish system.
  - Normal response code: 200 OK
  - Example response:

```
{
    "@odata.type": "#ServiceRoot.v1_0_0.ServiceRoot",
    "Id": "IronicProxy",
```

(continues on next page)

(continued from previous page)

```
    "Name": "Ironic Redfish Proxy",
    "RedfishVersion": "1.0.0",
    "Links": {
        "Sessions": {
            "@odata.id": "/redfish/v1/SessionService/Sessions"
        }
    },
    "Systems": {
        "@odata.id": "/redfish/v1/Systems"
    },
    "SessionService": {
        "@odata.id": "/redfish/v1/SessionService"
    },
    "@odata.id": "/redfish/v1/"
}
```

Name	Type	Description
@odata.type	string	The type of the emulated Redfish resource.
@odata.id	string	A resource link.
Id	string	The identifier for this specific resource.
Name	string	The name of this specific ServiceRoot.
Links	object	Contains objects that contain links to relevant resource collections.
Systems	object	Contains a link to a collection of Systems resources.
SessionService	object	Contains a link to the SessionsService resource.
Sessions	object	Contains a link to a collection of Sessions resources.
RedfishVersion	string	The version of this Redfish service.

Sessions

- GET /redfish/v1/SessionService
  - Returns a Redfish SessionService object, containing information about how the SessionService and Session objects are configured.
    - If the underlying Ironic system is using HTTP basic auth, the SessionService will report itself to be disabled, and all Session- related functionality will be non-functional.
  - Normal response code: 200 OK
  - Error response codes: 404 Not Found, 500 Internal Server Error
    - 404 Not Found will be returned if the underlying Ironic system is not using Keystone authentication.
    - 500 Internal Server Error will be returned if the internal request to authenticate could not be fulfilled.

  - Example response:


```
{
    "@odata.type": "#SessionService.v1_0_0.SessionService",
    "Id": "KeystoneAuthProxy",
    "Name": "Redfish Proxy for Keystone Authentication",
    "Status": {
        "State": "Enabled",
        "Health": "OK"
    },
    "ServiceEnabled": true,
    "SessionTimeout": 86400,
    "Sessions": {
        "@odata.id": "/redfish/v1/SessionService/Sessions"
    },
    "@odata.id": "/redfish/v1/SessionService"
}
```


Name	Type	Description
@odata.type	string	The type of the emulated Redfish resource.
@odata.id	string	A resource link.
Id	string	The identifier for this specific resource.
Name	string	The name of this specific resource.
Status	object	An object containing service status info.
State	string	The state of the service, one of either Enabled or Disabled.
Health	string	The health of the service, typically OK.[1]
ServiceEnabled	bool	Indicates whether the SessionService is enabled or not.
SessionTimeout	number	The amount of time, in seconds, before a session expires due to inactivity.[2]
Sessions	object	Contains a link to a collection of Session resources.

- GET /redfish/v1/SessionService/Sessions
  - Returns a Redfish SessionCollection, containing a link to the Session being used to authenticate the request. Requires the user to provide valid authentication in the request header.
  - Normal response code: 200 OK
  - Error response codes: 401 Unauthorized, 404 Not Found, 500 Internal Server Error
    * 401 Unauthorized will be returned if authentication in the header field is either absent or invalid.
    * 404 Not Found will be returned if the underlying Ironic system is not using Keystone authentication.
    * 500 Internal Server Error will be returned if the internal request to authenticate could not be fulfilled.

[1] This is included for compatibility and should always be OK, although the Redfish schema allows for Warning and Critical as well.

[2] This is a placeholder value. Since sessions are just Keystone auth tokens, they will behave as any other Keystone token, as opposed to behaving like a Redfish Session. (see Authentication)

- Example response:

```
{
    "@odata.type": "#SessionCollection.SessionCollection",
    "Name": "Ironic Proxy Session Collection",
    "Members@odata.count": 1,
    "Members": [
        {
            "@odata.id": "/redfish/v1/SessionService/Sessions/ABC"
        }
    ],
    "@odata.id": "/redfish/v1/SessionService/Sessions"
}
```

Name	Type	Description
@odata.type	string	The type of the emulated Redfish resource.
@odata.id	string	A resource link.
Name	string	The name of this specific resource.
Members@odata.count	number	The number of Session interfaces present in the collection.
Members	array	An array of objects that contain links to individual Session interfaces.

- POST /redfish/v1/SessionService/Sessions
  - Requests Session authentication. A username and password is to be passed in the body, and upon success, the created Session object will be returned. Included in the headers of this response will be the authentication token in the `X-Auth-Token` header, and the link to the Session object in the `Location` header.
  - Normal response code: 201 Created
  - Error response codes: 400 Bad Request, 401 Unauthorized, 404 Not Found, 500 Internal Server Error
    - 400 Bad Request will be returned if the username/password fields are not present in the message body.
    - 401 Unauthorized will be returned if the credentials provided are invalid.
    - 404 Not Found will be returned if the underlying Ironic system is not using Keystone authentication.
    - 500 Internal Server Error will be returned if the internal request to authenticate could not be fulfilled.
  - Example Request:

```
{
    "UserName": "85775665-c110-4b85-8989-e6162170b3ec",
    "Password": "its-a-secret-shhhhh"
}
```

Name	Type	Description
User-Name	string	The UUID of the Keystone application credential to be used for authentication.
Pass-word	string	The secret of said application credential.

- Example Response:

```
Location: /redfish/v1/SessionService/Sessions/identifier
X-Auth-Token: super-duper-secret-aaaaaaaaaaaa

{
    "@odata.id": "/redfish/v1/SessionService/Sessions/identifier",
    "@odata.type": "#Session.1.0.0.Session",
    "Id": "identifier",
    "Name": "user session",
    "UserName": "85775665-c110-4b85-8989-e6162170b3ec"
}
```

Name	Type	Description
@odata.type	string	The type of the emulated Redfish resource.
@odata.id	string	A resource link.
Id	string	The identifier for this specific resource.
Name	string	The name of this specific resource.
UserName	string	The UUID of the application credential used for authentication.

- GET /redfish/v1/SessionService/Sessions/{identifier}
  - Returns the Session with the identifier specified in the URL. Requires the user to provide valid authentication in the request header for the session theyre attempting to access.
  - Normal response code: 200 OK
  - Error response codes: 401 Unauthorized, 403 Forbidden, 404 Not Found, 500 Internal Server Error
    - 401 Unauthorized will be returned if authentication in the header field is either absent or invalid.
    - 403 Forbidden will be returned if authentication in the header field is valid but lacking proper authorization for the Session being accessed.
    - 404 Not Found will be returned if the identifier specified does not correspond to a legitimate Session ID or if the underlying Ironic system is not using Keystone authentication.
    - 500 Internal Server Error will be returned if the internal request to authenticate could not be fulfilled.
  - Example Response:

```
{
    "@odata.id": "/redfish/v1/SessionService/Sessions/identifier",
    "@odata.type": "#Session.1.0.0.Session",
    "Id": "identifier",
    "Name": "user session",
    "UserName": "85775665-c110-4b85-8989-e6162170b3ec"
}
```

Name	Type	Description
@odata.type	string	The type of the emulated Redfish resource.
@odata.id	string	A resource link.
Id	string	The identifier for this specific resource.
Name	string	The name of this specific resource.
UserName	string	The application credential used for authentication

- DELETE /redfish/v1/SessionService/Sessions/{identifier}
  - Ends the session identified in the URL. Requires the user to provide valid authentication in the request header for the session theyre trying to end.
  - Normal response code: 204 No Content
  - Error response codes: 401 Unauthorized, 403 Forbidden, 404 Not Found, 500 Internal Server Error
    - 401 Unauthorized will be returned if authentication in the header field is either absent or invalid.
    - 403 Forbidden will be returned if authentication in the header field is valid but lacking proper authorization for the Session being accessed.
    - 404 Not Found will be returned if the identifier specified does not correspond to a legitimate Session ID or if the underlying Ironic system is not using Keystone authentication.
    - 500 Internal Server Error will be returned if the internal request to authenticate could not be fulfilled.

Node Management

- GET /redfish/v1/Systems
  - Equivalent to `baremetal node list`, will return a collection of Redfish ComputerSystem interfaces that correspond to Ironic nodes. Requires the user to provide valid authentication in the request header for the resource they are trying to access.
  - Normal response code: 200 OK
  - Error response codes: 401 Unauthorized, 403 Forbidden, 500 Internal Server Error
    - 401 Unauthorized will be returned if the authentication in the header field is either absent or invalid.
    - 403 Forbidden will be returned if authentication in the header field is valid but lacking proper privileges for listing Bare Metal nodes.

    * 500 Internal Server Error will be returned if the internal request to the Bare Metal service could not be fulfilled.

  - Example Response:

```
{
    "@odata.type": "#ComputerSystemCollection.
↪ComputerSystemCollection",
    "Name": "Ironic Node Collection",
    "Members@odata.count": 2,
    "Members": [
        {
            "@odata.id": "/redfish/v1/Systems/ABCDEFG"
        },
        {
            "@odata.id": "/redfish/v1/Systems/HIJKLMNOP"
        }
    ],
    "@odata.id": "/redfish/v1/Systems"
}
```

Name	Type	Description
@odata.type	string	The type of the emulated Redfish resource.
@odata.id	string	A resource link.
Name	string	The name of this specific resource.
Members@odata.count	number	The number of System interfaces present in the collection.
Members	array	An array of objects that contain links to individual System interfaces.

- GET /redfish/v1/Systems/{node_ident}
  - Equivalent to `baremetal node show`, albeit with fewer details. Will return a Redfish System resource containing basic info, power info, and the location of the power control interface. Requires the user to provide valid authentication for the resource they are trying to access.
  - Normal response code: 200 OK
  - Error response codes: 401 Unauthorized, 403 Forbidden, 404 Not Found, 500 Internal Server Error
    * 401 Unauthorized will be returned if the authentication in the header field is either absent or invalid.
    * 403 Forbidden will be returned if authentication in the header field is valid but lacking proper privileges for the Bare Metal node being accessed.
    * 404 Not Found will be returned if the identifier specified does not correspond to a legitimate node UUID.
    * 500 Internal Server Error will be returned if the internal request to the Bare Metal service could not be fulfilled.
  - Example Response:

```
{
    "@odata.type": "#ComputerSystem.v1.0.0.ComputerSystem",
    "Id": "ABCDEFG",
    "Name": "Baremetal Host ABC",
    "Description": "It's a computer",
    "UUID": "ABCDEFG",
    "PowerState": "On",
    "Actions": {
        "#ComputerSystem.Reset": {
            "target": "/redfish/v1/Systems/ABCDEFG/Actions/
↪ComputerSystem.Reset",
            "ResetType@Redfish.AllowableValues": [
                "On",
                "ForceOn",
                "ForceOff",
                "ForceRestart",
                "GracefulRestart",
                "GracefulShutdown"
            ]
        }
    },
    "@odata.id": "/redfish/v1/Systems/ABCDEFG"
}
```

Name	Type	Description
@odata.type	string	The type of the emulated Redfish resource.
@odata.id	string	A resource link.
Id	string	The identifier for this specific resource. Equal to the corresponding Ironic node UUID.
Name	string	The name of this specific resource. Equal to the name of the corresponding Ironic node if set, otherwise equal to the node UUID.
Description	string	If the Ironic node has a description set, it will be returned here. If not, this field will not be returned.
UUID	string	The UUID of this resource.
PowerState	string	The current state of the node/System in question, one of either On, Off, Powering On, or Powering Off.
Actions	object	Contains the defined actions that can be executed on this system.
#ComputerSystem. Reset	object	Contains information about the Reset action.
target	string	The URI of the Reset action interface.
ResetType@Redfish. AllowableValues	array	An array of strings containing all the valid options this action provides.

- POST /redfish/v1/Systems/{node_ident}/Actions/ComputerSystem.Reset
  - Invokes a Reset action to change the power state of the node/System. The type of Reset

action to take should be specified in the request body. Requires the user to provide valid authentication in the request header for the resource they are attempting to access.

- Accepts the following values for ResetType in the body[3]:
  - On (soft power on)
  - ForceOn (hard power on)
  - GracefulShutdown (soft power off)
  - ForceOff (hard power off)
  - GracefulRestart (soft reboot)
  - ForceRestart (hard reboot)
- Normal response code: 202 Accepted
- Error response codes: 400 Bad Request, 401 Unauthorized, 403 Forbidden, 404 Not Found, 409 NodeLocked/ClientError, 500 Internal Server Error, 503 NoFreeConductorWorkers (for more on codes 409 and 503, see the details for PUT requests to `/v1/nodes/{ident}/states/power` in [IRONCAPI]):
  - 400 Bad Request will be returned if the ResetType field is not found in the message body, or if the field has an invalid value.
  - 401 Unauthorized will be returned if the authentication in the header field is either absent or invalid.
  - 403 Forbidden will be returned if authentication in the header field is valid but lacking proper privileges to perform the specified action on the Bare Metal node being accessed.
  - 404 Not Found will be returned if the identifier specified does not correspond to a legitimate node UUID.
  - 409 NodeLocked/ClientError is an error code specified in the Bare Metal API call this request is proxied to. The body of a 409 response will be the same as that which was received from the Bare Metal API.
  - 500 Internal Server Error will be returned if the internal request to the Bare Metal service could not be fulfilled.
  - 503 NoFreeConductorWorkers is an error code specified in the Bare Metal API call this request is proxied to. The body of a 503 response will be the same as that which was received from the Bare Metal API.
- Example Request:

```
X-Auth-Token: super-duper-secret-aaaaaaaaaaaa

{
    "ResetType": "ForceOff"
}
```

[3] The Redfish schema for ResetType also includes Nmi (diagnostic interrupt) and PushPowerButton (simulates a physical power button press event). However, neither of these actions are part of Ironics node power state workflow and support for these actions varies greatly depending on the driver and hardware.

Name	Type	Description
ResetType	string	The type of Reset action to take (see above)

Client (CLI) impact

None.

openstack baremetal CLI

Though this addition would include new REST API endpoints, this feature merely provides another way for users to access already existing features within the Ironic API, which are already accessible from the `openstack baremetal` CLI.

openstacksdk

None.

RPC API impact

None.

Driver API impact

None.

Nova driver impact

None.

Ramdisk impact

None.

Security impact

The main consideration when it comes to the security of this feature is the addition of a new means of accessing Ironic hardware. However, this should not pose much of a new concern when it comes to security, since authentication shall be implemented in the same way and using the same midddlewares as the existing Ironic API. Nevertheless, great care will be taken to make sure that said integration with the existing auth middleware is safe and secure.

To mitigate further risk, generated application credentials can and should be limited in scope to only allow access to the parts of the Ironic API required by this intermediary. We will also require all requests to be performed over HTTPS, since session tokens, application credential secrets, and (base64- encoded) HTTP basic auth credentials will be sent in plain text.

Other end user impact

This will give end users an alternative way of accessing power controls, one compatible with existing Redfish provisioning tools. This means in theory, the majority of users wont be making API calls directly, instead utilizing pre-existing Redfish-compatible software, such as Redfishtool.

Scalability impact

None.

Performance Impact

Since this shall be implemented as a separate WSGI service, some additional overhead will be required, although the impact should be minor, as it will not require the running of any periodic tasks nor the execution of any extraneous database queries.

Additionally, it should be noted that running this proxy service is completely optional on the part of the Ironic system operator; if one does not wish to use it, its existence can simply be ignored.

Other deployer impact

This feature should have no impact on those who do not wish to use it, as it must be ran separately and can be ignored. To prevent it from being started accidentally, operators shall be able to explicitly disable it in the Ironic configuration file.

Those who wish to make use of this feature must keep in mind that since it is currently being implemented as a separate WSGI service, it shall require at minimum its own port to be ran on. This can be useful if one wishes to have the Redfish proxy service bound to a different port or a different host IP from the Ironic API; however, it will require a new endpoint to be added via Keystone (if using Keystone), and may potentially require extra network configuration on the part of the system administrator.

Developer impact

This new service shall be implemented in Flask, as opposed to Pecan, which is what the Ironic API currently uses. As such, all code written for this new feature shall be well-documented in order to maximize its readability to any Ironic devs unfamiliar with Flask.

It has been mentioned by TheJulia that in future, the Ironic dev team may want to look into migrating the Ironic API to use Flask, and this addition to the codebase may prove useful to those tasked with said migration.

Finally, the Sessions feature does not exist in sushy-tools; since this spec includes a planned implementation of it, it could possibly be a useful addition there. This basic Redfish proxy can also be extended in future to provide access to even more parts of an Ironic system through a Redfish-like interface to those who would find such functionality useful.

Implementation

Assignee(s)

Primary assignee:

Sam Zuk (sam_z / szuk) <szuk@redhat.com>

Other contributors:

Tzu-Mainn Chen (tzumainn) <tzumainn@redhat.com>

Work Items

- Create the necessary API endpoints
  - Implement the Redfish System -> Ironic Node proxy
  - Implement the Redfish Session -> Keystone authentication proxy
  - Write unit tests and functional tests to ensure proper functionality
- Write documentation for how to use and configure this functionality, for users, administrators, and developers.
- Test this feature on real hardware in a way that mimics expected use cases.

Dependencies

None.

Testing

Functional testing will be required to ensure requests made to these new proxy endpoints result in the correct behavior when ran on an actual Ironic setup. Furthermore, rigorous test cases should be written to make extremely sure that no unauthorized access to node APIs is possible.

Upgrades and Backwards Compatibility

N/A

Documentation Impact

Documentation will need to be provided for the new API endpoints, along with the necessary instructions for how to enable and configure this feature (for operators), along with additional information end users may require, such as how to work with authentication tokens.

References

5.15.158 Implementation of ironic commands as an OSC plugin

https://bugs.launchpad.net/ironic/+bug/1526479

The OpenStackClient is becoming the defacto CLI client for OpenStack. This spec will spell out what the command structure should look like, including parameters and command names.

Problem description

The OpenStackClient[1] has become the preferred method of creating clients for OpenStack APIs. The initial implementation has been done. What needs to happen is to define what the command structure should be. There has been some confusion/discussion[2] about what these commands should look like, so it seemed the proper thing to create a spec.

The goal of the OpenStackClient is to make the CLI clients more natural for the End User. This spec will specify the commands that the End User will use when interacting with Ironic.

[1] http://docs.openstack.org/developer/python-openstackclient/index.html
[2] http://lists.openstack.org/pipermail/openstack-dev/2015-November/078998.html

Proposed change

The proposed implementation will have all of the commands implemented as specified in the *Client (CLI) impact* section below.

In addition (or as clarification) to the OpenStackClient command structure[3] :

- the OpenStackClient command structure is described as <object1> <action> <object2>. This doesnt work if there are commands of the form <object1> <action>. Instead, we will use the form <object1> <object2> <action>. (Perhaps think of it as one object with two parts). For example, instead of openstack baremetal node set maintenance (because we have openstack baremetal node set), we will use openstack baremetal node maintenance set.
- dont use hyphenated nouns, because the commands should be more natural and there arent any commands (yet) that use hyphens. For example, instead of openstack baremetal node boot-device set, we are going to use openstack baremetal node boot device set.
- only provide one OpenStackClient command to do something; avoid aliasing
- for naming, the trend is to use Americanised spelling, eg favor instead of favour. Having said that, it is important to take into consideration the terminology/usage outside of OpenStack, e.g. by operators and administrators.

Alternatives

Continue with the current client and remove the existing OSC plugin bits.

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

openstack baremetal chassis

- openstack baremetal chassis show <uuid>

  --fields <field,[field,]> Select fields to fetch and display

- openstack baremetal chassis list

  --long Show detailed chassis info (Mutually exclusive to fields)

  --limit <limit> Limit the number of items to return

  --marker <uuid> Which chassis uuid to start after

[3] http://docs.openstack.org/developer/python-openstackclient/commands.html

--sort <key[:direction]> Key and direction of sort. <direction> is optional. Defaults to ascending order.

--fields <field,[field,]> Select fields to fetch and display. (Mutually exclusive to long)

- openstack baremetal chassis create

  --description <description> Chassis description

  --extra <key=value> Extra chassis properties. Can be specified multiple times.

  --uuid <uuid> UUID of the chassis

- openstack baremetal chassis delete <uuid> [<uuid>]
- openstack baremetal chassis set <uuid>

  --extra <key=value> Property to set or update. Can be specified multiple times.

  --description <description> Chassis description

- openstack baremetal chassis unset <uuid>

  --extra <key> Key of property to unset. Can be specified multiple times.

  --description <description> Will unset the chassis description ()

ironic CLI users who want to see a list of nodes belonging to a given chassis should use `openstack baremetal node list --chassis`, since we will not provide an `openstack baremetal chassis xxx` equivalent to `ironic chassis-node-list`.

openstack baremetal driver

- openstack baremetal driver list
- openstack baremetal driver show <driver>
- openstack baremetal driver passthru list <driver>
- openstack baremetal driver passthru call <driver> <method>

  <method> Vendor passthru method to call.

  --arg <key=value> key=value to add to passthru method. Can be specified multiple times.

  --http-method <http_method> one of POST, PUT, GET, DELETE, PATCH

openstack baremetal node

- openstack baremetal node show <uuid>

  Obsoletes: openstack baremetal show

  --instance Interpret <uuid> as an instance UUID

  --fields <field,[field,]> Select fields to fetch and display.

- openstack baremetal node list

  Obsoletes: openstack baremetal list

  --limit <limit> Limit the number of items to return

--marker <uuid> Which node to start after

--sort <key[:direction]> Key and direction of sort. <direction> is optional.

--maintenance List nodes in maintenance mode

--associated List nodes associated with an instance

--chassis <uuid> UUID of chassis to limit node list

--provision-state <state> Show nodes in specified <state>

--fields <field,[field,]> Select fields to fetch and display. (Mutually exclusive to long)

- openstack baremetal node create

  Obsoletes: openstack baremetal create

  --chassis-uuid <uuid> Chassis this node belongs to

  --driver <driver> Driver used to control the node

  --driver-info <key=value> key=value pair used by the driver. Can be specified multiple times.

  --property <key=value> Property of the node. Can be specified multiple times.

  --extra <key=value> Arbitrary metadata. Can be specified multiple times.

  --uuid <uuid> Unique UUID of the node. Optional.

  --name <name> Unique name of the node.

- openstack baremetal node delete <uuid> [<uuid>]

  Obsoletes: openstack baremetal delete

- openstack baremetal node set <uuid>

  Obsoletes: openstack baremetal set

  --name <name> Name of the node

  --instance-uuid <uuid> Instance UUID

  --driver <driverid> Driver name or UUID

  --property <key=value> Property to set/update on the node. Can be specified multiple times.

  --extra <key=value> Extra to set/update on the node. Can be specified multiple times.

  --driver-info <key=value> driver-info to set/update on the node. Can be specified multiple times.

  --instance-info <key=value> instance-info to set/update on the node. Can be specified multiple times.

  --target-raid-config <config> Set the target RAID configuration (JSON) for the node. This can be one of: 1. a file containing JSON data of the RAID configuration; 2. - to read the contents from standard input; or 3. a valid JSON string.

- openstack baremetal node unset <uuid>

  Obsoletes: openstack baremetal unset

  --property <key> key to unset on the node. Can be specified multiple times.

  --extra <key> key from extra to unset. Can be specified multiple times.

  --driver-info <key> key to unset from driver-info. Can be specified multiple times.

  --instance-info <key> key to unset from instance-info. Can be specified multiple times.

  --instance-uuid <uuid> Instance uuid.

  --name Name of the node.

  --target-raid_config target RAID configuration

- openstack baremetal node passthru list <uuid>
- openstack baremetal node passthru call <uuid> <method>

  <method> Vendor passthru method to be called

  --arg <key=value> argument to send to passthru method. Can be specified multiple times.

  --http-method <http_method> One of POST, PUT, GET, DELETE, PATCH

- openstack baremetal node console show <uuid>
- openstack baremetal node console enable <uuid>
- openstack baremetal node console disable <uuid>
- openstack baremetal node boot device show <uuid>

  --supported Show the supported boot devices

- openstack baremetal node boot device set <uuid> <device>

  <device> One of pxe, disk, cdrom, bios, safe

  --persistent Make changes persistent for all future boots.

- openstack baremetal node deploy <uuid>

  --config-drive <config_drive> A gzipped, base64-encoded configuration drive string OR the path to the configuration drive file OR the path to a directory containing the config drive files. In case its a directory, a config drive will be generated from it.

- openstack baremetal node undeploy <uuid>
- openstack baremetal node rebuild <uuid>
- openstack baremetal node inspect <uuid>
- openstack baremetal node provide <uuid>
- openstack baremetal node manage <uuid>
- openstack baremetal node abort <uuid>
- openstack baremetal node maintenance set <uuid>

--reason <reason> Reason for setting to maintenance mode

- openstack baremetal node maintenance unset <uuid>
- openstack baremetal node power on <uuid>
- openstack baremetal node power off <uuid>
- openstack baremetal node reboot <uuid>
- openstack baremetal node validate <uuid>

ironic CLI users who want to see a list of ports belonging to a given node should use `openstack baremetal port list --node`, since we will not provide an `openstack baremetal node xxx` equivalent to `ironic node-port-list`.

ironic CLI users who want the equivalent to `ironic node-show-states` should use the following command:

```
openstack baremetal node show <node> --fields console_enabled last_error
power_state provision_state provision_updated_at raid_config
target_power_state target_provision_state target_raid_config
```

openstack baremetal port

- openstack baremetal port show <uuid|mac>

  --address <mac> Mac address instead of uuid

  --fields <field[,field,]> Fields to display

- openstack baremetal port list

  --limit <limit> Limit the number of items to return

  --marker <marker> Which port to start after

  --sort <key[:direction]> Key and direction of sort

  --long Display detailed information about ports. Mutually exclusive with fields.

  --fields <field[,field,]> Fields to display. Mutually exclusive with long.

  --node <nodeid> UUID or name of node to limit the port display

- openstack baremetal port create <address>

  --node <uuid> Node uuid to add the port to

  --extra <key=value> Arbitrary key=value metadata. Can be specified multiple times.

- openstack baremetal port delete <uuid> [<uuid>]
- openstack baremetal port set <uuid>

  --extra <key=value> property to set. Can be specified multiple times.

  --address <macaddress> Set new MAC address of port

  --node <nodeid> Set UUID or name of node the port is assigned to

- openstack baremetal port unset <uuid>

--extra <key> key to remove. Can be specified multiple times.

Not addressed

OpenStackClient commands corresponding to these ironic CLI commands are not addressed by this proposal. They will be addressed in a future release.

- **`ironic driver-raid-logical-disk-properties`. Get RAID logical disk** properties for a driver.
- **`ironic driver-properties`. Get properties (node.driver_info keys and** descriptions) for a driver.

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

N/A

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

None

Developer impact

None

Implementation

Assignee(s)

Primary assignee:

- brad-9 <brad@redhat.com>

Other contributors:

- Romanenko_K <kromanenko@mirantis.com>
- rloo <ruby.loo@intel.com>

Work Items

TBD

Dependencies

None

Testing

Unittests will be added.

Upgrades and Backwards Compatibility

There is already an implementation of some of these commands. A few are likely to change with this spec. These existing commands will go through a deprecation period.

Documentation Impact

The command line documentation will be updated to show these new commands.

References

5.15.159 In-band cleaning support in iSCSI deploy drivers

https://bugs.launchpad.net/ironic/+bug/1526405

Current deploy drivers that make use of iSCSI dont support in-band cleaning. We need to make these drivers support in-band cleaning operations as well.

Problem description

Drivers that do iSCSI based deployment dont support in-band cleaning today. Hence in-band cleaning steps like disk erase, in-band RAID configuration, etc cannot be performed on these nodes which are registered with these drivers.

The following drivers (omitting the testing drivers) do iSCSI based deployment today:

- **pxe_{ipmitool,ipminative,seamicro,iboot,ilo,**
  drac,snmp,irmc,amt,msftocs,ucs,wol}

- iscsi_ilo
- iscsi_irmc

Proposed change

- Deprecate the opt `CONF.agent.agent_erase_devices_priority` and move that to `CONF.deploy.agent_erase_devices_priority`.
- Add the following methods to iscsi_deploy.ISCSIDeploy (these methods will do the same things as what AgentDeploy does today)
  - `prepare_cleaning` - This method will create the Neutron cleaning ports for each of the Ironic ports and will call `self.boot` to prepare for booting the ramdisk and return `states.CLEANWAIT`. The method definition will be as follows:

```
def prepare_cleaning(self, task):
    """Boot into the agent to prepare for cleaning.

    :param task: a TaskManager object containing the node
    :raises NodeCleaningFailure: if the previous cleaning ports␣
↪cannot
        be removed or if new cleaning ports cannot be created
    :returns: states.CLEANWAIT to signify an asynchronous prepare
    """
```

  - `tear_down_cleaning` - This method will delete the cleaning ports created for the Ironic ports and will call `self.boot` to clean up the ramdisk boot. It will return None. The method definition will be as follows:

```
def tear_down_cleaning(self, task):
    """Cleans up the environment after cleaning.

    :param task: a TaskManager object containing the node
    :raises NodeCleaningFailure: if the cleaning ports cannot be
        removed
    """
```

  - `execute_clean_step` - This method will call `deploy_utils.agent_execute_clean_step`. The method definition will be as follows:

```
def execute_clean_step(self, task, step):
    """Execute a clean step asynchronously on the agent.

    :param task: a TaskManager object containing the node
    :param step: a clean step dictionary to execute
    :raises: NodeCleaningFailure if the agent does not return a␣
↪command
        status
    :returns: states.CLEANWAIT to signify the step will be completed
        async
    """
```

  - `get_clean_steps` - This method will call `deploy_utils.agent_get_clean_steps` to get the cleaning steps from the agent ramdisk. It will also reassign the cleaning priority to disk erase.

    This will return an empty list if bash ramdisk is used. It will be detected by checking if `agent_url` is present in nodes driver_internal_info.

    The method definition will be as follows:

```
def get_clean_steps(self, task):
    """Get the list of clean steps from the agent.

    :param task: a TaskManager object containing the node
    :returns: A list of clean step dictionaries. Returns an
        empty list if bash ramdisk is used.
"""
```

- For deployers who have been using DIB ramdisk, the node will be stuck i n `states.CLEANWAIT` when they try to do cleaning. This is because DIB ramdisk doesnt heartbeat like agent ramdisk. Hence, such deployers might face the issue with nodes moving to `states.CLEANFAIL` as node enters cleaning. To overcome this problem, the following will be done:
  - Send the bash ramdisk parameters (deploy_key, iscsi_target_iqn, etc) while booting the deploy ramdisk for cleaning. It will enable bash ramdisk to invoke pass_deploy_info vendor passthru.
  - If node is in CLEANWAIT in pass_deploy_info vendor passthru, then we set the clean steps for the node and ask conductor to resume cleaning.
  - We also skip validation for pass_deploy_info vendor passthru if node is in CLEANWAIT state.

Alternatives

None.

Data model impact

None.

State Machine Impact

None.

REST API impact

None.

Client (CLI) impact

None.

RPC API impact

None.

Driver API impact

None.

Nova driver impact

None.

Ramdisk impact

N/A

Security impact

Drivers using `iscsi_deploy.ISCSIDeploy` will do in-band disk erase which will be a security benefit for tenants.

Other end user impact

None.

Scalability impact

None.

Performance Impact

None.

Other deployer impact

None.

Developer impact

None.

Implementation

Assignee(s)

Primary assignee:
rameshg87

Work Items

- Add new methods the `iscsi_deploy.ISCSIDeploy` for in-band cleaning.
- Modify pass_deploy_info to make it ready when it is invoked during cleaning.

Dependencies

- Completion of work for deploy-boot interface separation [2] to enable in-band cleaning for all drivers.

Testing

Unit tests will be added.

Upgrades and Backwards Compatibility

None.

Documentation Impact

The new CONF option and its impact will be documented.

References

[1] http://specs.openstack.org/openstack/ironic-specs/specs/approved/deprecate-bash-ramdisk.html [2] https://blueprints.launchpad.net/ironic/+spec/new-boot-interface

5.15.160 Kea DHCP backend

https://bugs.launchpad.net/ironic/+bug/2081847

Problem description

DHCP is a critical service for assigning IP addresses to instances in OpenStack deployments. Both Ironic and Neutron rely on dnsmasq to provide DHCP services. However, dnsmasq has become increasingly unreliable, with intermittent DHCP failures, frequent restarts and crashes of dnsmasq processes, etc.

Refer to the bug on dnsmasq here: https://bugs.launchpad.net/ironic/+bug/2026757.

Proposed change

Add a new Kea DHCP backend for Ironic and Neutron as an alternative DHCP provider. This will provide a more reliable DHCP option thats easily extensible while maintaining compatibility with existing systems.

- Create a new DHCP provider class extending the BaseDHCP interface.
- Add configuration options to enable and configure Kea DHCP.
- Add methods to manage DHCP options, leases, and port updates using Kea APIs.
- Ensure DevStack support for Kea
- Update documentation to cover the addition of Kea DHCP, setup and usage.

Ironic will interact with the Kea DHCP server via HTTP-based APIs[1].

- `config-get`, `config-set`: Manage Kea configuration
- `reservation-get`: Retrieve host reservations that will be handled externally via a host-reservation database shared between Kea and Ironic/Neutron query to retrieve reservation data.

[1] https://kea.readthedocs.io/en/latest/api.html

Dynamic reservation updates (`reservation-add` and `reservation-del`) require a premium ISC subscription and are not supported in this implementation[3].

- `lease4-get`, `lease6-get`: Retrieve lease information
- `subnet4-add`, `subnet6-add`: Add new subnets
- `statistic-get`: Monitor DHCP server statistics

The Kea DHCP backend can be added with or without an agent layer. Both approaches will use Keas HTTP-based APIs, but differ in their architecture and how Ironic interacts with the Kea DHCP server.

An agent approach has more moving parts and complexity, but better scalability for large deployments, local management of Kea instances and reduced network traffic to central Ironic service.

- Flow: Ironic Driver Agents Kea Servers
- Kea DHCP Agent is deployed on each DHCP server node and manages local Kea config and handles communication between Ironic and Kea.
- Ironic DHCP Driver implements Ironic DHCP interface and communicates with the Kea DHCP Agent.
- Ironic sends DHCP configuration to Ironic DHCP Driver which in turn distributes it to Kea agents to apply config to local Kea servers.
- For lease management, agents monitor local Kea lease changes report changes back to Ironic DHCP Driver.

The agentless approach will have a much simpler architecture, probably easier to implement and maintain with direct control from Ironic, but may not scale as well, increased network traffic to Kea servers and even a potential performance impact on Ironic for large-scale operations.

- **Flow: Ironic driver translates configs to Kea API calls and applies changes** directly to Kea servers.
- Ironic manages all Kea servers, through direct Kea API no agent layer.
- The Ironic driver translates configs to Kea API calls and applies changes directly to Kea servers.
- For lease management, maybe a periodic polling of Kea servers for lease updates, or, a webhook mechanism.

Alternatives

- Find a way to improve existing Dnsmasq implementations. Its worth noting that dnsmasq has inherent limitations as its really intended for small computer networks even though Ironic has been using it for the longest, so addressing these problems may require significant re-engineering. Its also single maintainer OSS project, as of the time of this specification. The fact it is a single maintainer project results in long spans of inactivity, which increases consumer risk for a project like OpenStack.
- Use a different DHCP provider than Kea that is also actively maintained, provides the reliability, flexibility, and integration Kea offers or better.
- Develop a completely new, custom DHCP server which will be quite an undertaking and additional long-term technical debt.

[3] https://kea.readthedocs.io/en/latest/arm/hooks.html

Data model impact

No changes to Ironics data model are required.

State Machine Impact

No changes to the state machine are required.

REST API impact

No changes to the REST API are required.

Client (CLI) impact

No changes to python-ironicclient are required.

RPC API impact

No changes to the RPC API are required.

Driver API impact

A new DHCP provider class will be added, but this should not impact existing drivers.

Nova driver impact

None

Ramdisk impact

No changes to the ironic-python-agent or ramdisk are required.

Security impact

Theres no expected security trade-offs with the addition of a Kea DHCP backend. The increase in options for operators limits overall risk by providing additional options, which should be a net security gain.

In the event of such occurrences in the future, its active development will likely ensure timely security updates.

Other end user impact

None

Scalability impact

Kea DHCP is designed for better scalability than dnsmasq, which could improve performance, especially for large deployments.

Performance Impact

Kea DHCP will likely offer performance improvements over dnsmasq, especially for large deployments with thousands of machines.

Other deployer impact

Deployers will need to install and configure the Kea DHCP server alongside Ironic, likely in the same manner as dnsmasq, but with Kea-specific configurables such as network interfaces, IP address ranges, and lease times.

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
Afonne-CID (cid).

Other contributors:
Jay Faulkner (JayF).

Work Items

- Write a Kea DHCP backend extending the BaseDHCP interface.
- Add unit tests and DevStack support for running with Kea.
- Configure at least one CI job to use Kea DHCP.
- Add Bifrost support for the new Kea DHCP backend.
- Implement unmanaged inspection support for Kea DHCP.
- Update documentation.

Dependencies

- Kea DHCP server[2].

Testing

Add tests to verify full DevStack support, new Tempest tests specific to Kea DHCP functionality, and new unit tests for: * Retrieving lease information via Kea APIs. * Ensuring Kea correctly reads from the host-reservation database.

Upgrades and Backwards Compatibility

Existing dnsmasq support will remain unchanged.

[2] https://www.isc.org/kea/

Documentation Impact

- Full documentation on Keas capabilities and how its different from dnsmasq with cross-references to external Kea resources.
- Document configuration options and steps on switching from dnsmasq to Kea, including configuring Kea to use a read-only host-reservation database and how to set up Ironic/Neutron to manage this database.
- Installation, configuration, and architecture documentations should present Kea as a configurable option, with clear instructions on how users can choose between Kea and dnsmasq.
- API documentation will need to be updated to reflect any changes to existing methods and how they now interact with Kea compared to dnsmasq.
- Sections of the current documentation that might have referenced dnsmasq as the default or only DHCP provider will need to be updated to reflect that Kea is now also a supported backend.

References

5.15.161 Keystone Policy Support

https://bugs.launchpad.net/ironic/+bug/1526752

Keystone and oslo.policy have support for restricting access based on information about the user authenticating, allowing partial access to be granted as configured by operators. This spec lays out how this support will be implemented in ironic.

Problem description

Ironic has traditionally operated on an all-or-nothing access system, only restricting access to passwords. This model is significantly limited when multiple people and groups with different trust levels want to interact with ironic. For example, a hardware technician may need access to set or unset maintenance on the node, but should not have access to provision nodes.

Proposed change

- Ensure proper metadata, such as role, is derived from the auth_token when authenticating by properly implementing KeystoneMiddleware.auth_token support.
- Define policy rules for each RESTful action on each API endpoint, scoped to the baremetal namespace.
- Configure each API endpoint to verify a user is permitted by policy to access it.
- Implement specific restrictions for sensitive information, including configdrives and passwords. Default to hide all sensitive information.
- Define sane default policies in code[0] , with shipped roles including an admin role with full access and an observer role with read-only access to non-secret information. Names for these roles will be determined during implementation. A sample policy.json[1] shall be generatable using oslopolicy-sample-generator.
- Maintain compatibility with all roles in the previously-shipped policy.json configuration file.

[0] Oslo Policy in Code https://specs.openstack.org/openstack/oslo-specs/specs/newton/policy-in-code.html
[1] Policy JSON syntax http://docs.openstack.org/kilo/config-reference/content/policy-json-file.html

Alternatives

A deployer could implement ironic behind a reverse proxy and use another authentication method to allow or disallow access based on path and HTTP method. This is onerous, does not follow the pattern set by other OpenStack services, and does not provide the granularity that properly implementing policy support would.

Data model impact

None.

State Machine Impact

Users may be restricted by policy from moving nodes within the state machine. However, there are no direct state machine modifications.

REST API impact

A properly restricted user may receive a 403 error if they are unable to use the method/endpoint combination requested. However, the REST API will not be returning 403 in any case it could not today, for instance, an unauthorized user may receive 403 today. This simply increases the granularity available for configuring this authorization.

The 403 response body shall indicate which resource access was denied to.

Client (CLI) impact

A CLI client user will need to have a properly authorized user to perform any requested actions.

RPC API impact

None.

Driver API impact

Drivers can now enforce policy within any driver_vendor_passthru methods as desired.

Nova driver impact

Existing deployments can continue to use a full-admin user as required prior to this feature. Once upgraded, a deployer could use a less-privileged user for nova-ironic interactions.

Ramdisk impact

N/A

Security impact

This changes primary impact is around improving the security of the system. Deployers of ironic will no longer need to provide an admin credential to manipulate only a small part of ironics API.

Other end user impact

None.

Scalability impact

None.

Performance Impact

Policy support is a minimal increase in overhead. Additionally, most policies will be implemented early in the API layer, to prevent ironic from doing excessive work before a user is deemed unauthorized.

Other deployer impact

Deployers will now be able to configure policies, in the policy.json DSL[Page 1248, 1] , to meet their specific needs.

Developer impact

Whenever a developer implements a new API method, they will be required to add a new policy rule to represent that API endpoint or method, define the default rule, enforce the policy appropriately, and update default policy as necessary.

Implementation

Assignee(s)

Primary assignee:
devananda

Other contributors:
JayF

Work Items

- Update devstack to configure users properly.
- Change configuration of nova in devstack to use new baremetal_driver role.
- Document how to utilize policy, including how to create users in keystone and assign them to the baremetal project.
- Document any differences in how this impacts users of Keystone API v2 vs v3.

Dependencies

None.

Testing

- Grenade testing to ensure we do not break existing deployments.
- Unit testing to ensure policies are being properly enforced.

Upgrades and Backwards Compatibility

Existing deployers are required to use an admin user for all uses of ironic, these users will continue to have full access to the ironic API, allowing for backwards compatibility.

On upgrade, an operator must define new keystone roles and assign these to users in order to take advantage of the new policy support. The names for these roles will be determined during implementation.

The operator may choose to customize the policy settings for their deployment.

Documentation Impact

- Default policies will need to be documented.
- Install guide will need to be updated with instructions on how to create users with proper roles and project membership.
- Documentation must be written instructing users how to utilize the new policy functionality on upgrade.

References

5.15.162 Manual cleaning

https://bugs.launchpad.net/ironic/+bug/1526290

Manual cleaning (as opposed to automated cleaning) encompasses all long running, manual, destructive tasks an operator may want to perform either between workloads, or before the first workload has been assigned to a node.

This feature had previously been called Zapping and this specification copies a lot of the zapping specification. (Thank you Josh Gachnang!)

Problem description

Automated cleaning has been available in ironic since the kilo cycle. It lets operators choose which clean steps are automatically done prior to the first time a node is deployed and each time after a node is released.

However, operators may want certain operations or tasks to only run on demand, rather than in every clean cycle. Things like firmware updates, setting up new RAID levels, or burning in nodes often need to be done before a user is given a server, but take too long to reasonably do at deploy time.

Many of the above tasks could provide useful scheduling hints to nova once hardware capabilities are introduced. Operators could use these scheduling hints to create flavors, such as a nova compute flavor that requires a node with RAID 1 for extra durability.

Proposed change

Instead of adding new ZAP* states to the state machine to distinguish between manual and automated cleaning, the existing CLEAN* states and cleaning mechanism will be reused for both automated and manual cleaning. The main differences will be:

- manual cleaning can only be initiated when a node is in the MANAGEABLE state. Once the manual cleaning is finished, the node will be put in the MANAGEABLE state again.
- operators will be able to initiate a manual clean via the modified API to set the nodess provision state. Details are described in the *PUT /states/provision* section below.

- A manual clean step might need some arguments to be specified. (This might be useful for future automated steps too.) To support this, the ironic.drivers.base.clean_step decorator will be modified to accept a list of arguments. (Default is None.) Each argument is a dictionary with:
  - name: <name of argument>
  - description: <description>. This should include possible values.
  - required: Boolean. True if this argument is required it must be specified in the manual clean request; false if it is optional.
- add clean steps to drivers that will only be used by manual cleaning. The mechanism for doing this exists already. Driver implementers only need to use the @clean_step decorator with a default cleaning priority of 0. This will ensure the step isnt run as part of the automated cleaning. The implementer can specify whether the step is abortable, and should also include any arguments that can be passed to the clean step.
- operators will be able to get a list of possible steps via an API. The *GET /cleaning/steps* section below provides more information.
- similar to executing automated clean steps, when the conductor attempts to execute a manual clean step, it will call execute_clean_step() on the driver responsible for that clean step.
- to avoid confusion, the clean_nodes config will be renamed to automated_clean_enable since it only pertains to automated cleaning. The deprecation and deletion of the clean_nodes config will follow ironics normal deprecation process.

Alternatives

- We could make manual clean steps and automated clean steps mutually exclusive with separate APIs and terminology and mechanisms to use, but conceptually, since they are all clean steps it is less confusing to provide a similar mechanism for both.
- We could have called manual clean something else like zap to avoid having to distinguish between manual and automated cleaning, but it seems more confusing to describe the differences between zap and clean and that confusion and complexity is apparent when trying to implement it that way.

Data model impact

None.

State Machine Impact

This:

- removes all mention of zap and the ZAP* states from the proposed state machine
- adds two new transitions:
  - MANAGEABLE -> CLEANING via clean verb, to start manual cleaning
  - CLEANING -> MANAGEABLE via manage verb, to end a successful manual clean

REST API impact

PUT /v1/nodes/<node_ident>/states/provision

This API will allow users to put a node directly into CLEANING provision state from MANAGEABLE state via target: clean. The PUT will also require the argument clean_steps to be specified. This is an ordered list of clean steps, with a clean step being represented as a dictionary encoded as JSON.

As an example:

```
'clean_steps': [{
    'interface': 'raid'
    'step': 'create_configuration',
    'args': {'create_nonroot_volumes': False, // optional keyword argument
             ... }               // more keyword arguments (if applicable)
  },
  {
    'interface': 'deploy'
    'step': 'erase_devices'
  }
]
```

In the above example, the drivers RAID interface would configure hardware RAID without non-root volumes, and then all devices would be erased (in that order).

A clean step is represented by a dictionary (JSON), in the form:

```
{
    'interface': <interface>,
    'step': <name of clean step>,
    'args': {<arg1>: <value1>, ..., <argn>: <valuen>}
}
```

The interface and step keys are required for all steps. If a step takes additional keyword arguments, the args key may be specified. It is a dictionary of keyword arguments, with each keyword-argument entry being <name>: <value>.

If any step is missing a required keyword argument, no manual cleaning will be performed and the node will be put in CLEANFAIL provision state with an appropriate error message.

If, during the cleaning process, a clean step determines that it has incorrect keyword arguments, all earlier steps will be performed and then the node will be put in CLEANFAIL provision state with an appropriate error message.

A new API version is needed to support this.

GET /nodes/<node_ident>/cleaning/steps

We had planned on having an API endpoint to allow operators to see the clean steps for an automated cleaning. That proposed API had been GET /nodes/<node_ident>/cleaning/clean_steps, but it hasnt been implemented yet.

With the introduction of manual cleaning, instead of GET /nodes/<node_ident>/cleaning/clean_steps, this proposes replacing that with the API endpoint GET /nodes/<node_ident>/cleaning/steps. By default, it will return all available clean steps (with priorities of zero and non-zero), for both manual and

automated cleaning.

An optional field min_priority can be specified to filter for clean steps with priorities equal to or above the specified minimum value. For example, to only get clean steps for automated cleaning (not manual):

```
GET http://127.0.0.1:6385/v1/nodes/my-awesome-node/cleaning/steps?min_
↪priority=1
```

The response to this request would be a list of clean steps sorted in decreasing priorities, formatted as follows:

```
[{
  // 'interface': is one of 'power', 'management', 'deploy', 'raid'.
  // 'step': is an opaque identifier used by the driver. Could be a driver
  //         function name or some function in the agent.
  // 'priority': is the priority used for determining when to execute
  //             the step; larger values have higher priority.
  // 'abortable': True if cleaning can be aborted during execution of this
  //              step; False otherwise.
  'interface': 'interface',
  'step': 'step',
  'priority': Integer,
  'abortable': Boolean

  // 'args': a list of keyword arguments that may be included in the
  //         'PUT /v1/nodes/NNNN/states/provision' request when doing
  //         a manual clean. An argument is a dictionary with:
  //           - 'name': <name of argument>
  //           - 'description': <description>
  //           - 'required': Boolean. True if required; false if optional
  'args': []
 },
 ... more steps ...
]
```

An example with a single step:

```
[{
  'interface': 'raid',
  'step': 'create_configuration',
  'args': [{'name':'create_root_volume',
            'description':'Set to True (the default) to create root volume
                           specified in the node's target_raid_config. False
                           prevents the root volume from being created.',
            'required':False},
           {'name':'create_nonroot_volumes',
            'description':'Set to True (the default) to create non-root
                           volumes that may be specified in the node's
                           target_raid_config. False prevents non-root
                           volumes from being created.',
            'required':False}]
  'priority': 0,
```

(continues on next page)

(continued from previous page)

```
  'abortable': True
}]
```

If the driver interface cannot synchronously get the list of clean steps, for example, because a remote agent is used to determine available clean steps, then the driver MUST cache the list of clean steps from the most recent execution of said agent and return that. In the absence of such data, the driver MAY raise an error, which should be translated by the API service into:

- an HTTP 202
- a new (we created this) HTTP header Retry-Request-After, indicating to the client how long in seconds the client should wait to retry. A -1 indicates that it is unknown how long to wait. This might happen for example when the request is made when a node is in ENROLL state. At this point it is unknown when the remote agent will be available on the node for querying.
- a body with a message indicating that the data are not available yet.

If the driver interface can synchronously return the clean steps without relying on the hardware or a remote agent, it SHOULD do so, though it MAY also rely on the aforementioned caching mechanism.

A new API version is needed to support this.

Client (CLI) impact

ironic node-set-provision-state

A new argument called clean-steps will be added to the node-set-provision-state CLI. Its value is a JSON file which is read and the contents passed to the API. Thus, the file has the same format as what is passed to the API for clean steps.

If the input file is specified as -, the CLI will read in from stdin, to allow piping in the clean steps. Using - to signify stdin is common in Unix utilities.

The clean-steps argument is required if the requested provision state target/verb is clean. Otherwise, specifying it is considered an error.

ironic node-get-clean-steps

A new node-get-clean-steps API will be added as follows:

```
ironic node-get-clean-steps [--min_priority <priority>] <node>

<node>: name or UUID of the node
--min-priority <priority>: optional minimum priority; default is 0 for all␣
↪clean steps
```

If successful, it will return a list of clean steps. If the response from the corresponding REST API request is an HTTP 202, it will return the message from that response body (that the data are not available) along with a suggestion to retry the request again.

RPC API impact

Add do_node_clean() (as a call()) to the RPC API and bump the RPC API version.

Driver API impact

None

Nova driver impact

None

Ramdisk impact

N/A

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

None

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
rloo (taking over from JoshNang who has left ironic)

Other contributors:
JoshNang (who started this)

Work Items

- Make the changes (as described above) to the state machine
- Bump API microversion to allow manual cleaning and implement the changes to PUT /v1/nodes/(node_ident)/states/provision API (as described above)
- Modify the cleaning flow to allow manual cleaning
- Change execute_clean_steps and get_clean_steps in any asynchronous driver to cache clean steps and return cached clean steps whenever possible.
- Allow APIs to return a Retry-Request-After HTTP header and empty response, in response to a certain exception from drivers.

Dependencies

- get_clean_steps API: https://review.opendev.org/#/c/159322

Testing

- Drivers implementing manual cleaning will be expected to test their added features.

Upgrades and Backwards Compatibility

None

Documentation Impact

The documentation will be updated to describe or clarify automated cleaning and manual cleaning and how to configure ironic to do one or both of them:

- http://docs.openstack.org/developer/ironic/deploy/install-guide.html
- http://docs.openstack.org/developer/ironic/deploy/cleaning.html
- http://docs.openstack.org/developer/ironic/webapi/v1.html will be updated to reflect the API version that supports manual cleaning

References

Automated cleaning specification: http://specs.openstack.org/openstack/ironic-specs/specs/kilo-implemented/implement-cleaning-states.html

State machine specification: http://specs.openstack.org/openstack/ironic-specs/specs/kilo-implemented/new-ironic-state-machine.html

Zapping related patches:

- Launchpad blueprint: https://blueprints.launchpad.net/ironic/+spec/implement-zapping-states
- **specification patches:**
  - https://review.opendev.org/#/c/185122/
  - https://review.opendev.org/#/c/209207/
- **code patches:**
  - https://review.opendev.org/#/c/221949/

- https://review.opendev.org/#/c/221989/
- https://review.opendev.org/#/c/223295/
- https://review.opendev.org/#/c/223311/

5.15.163 Project Mercury

https://bugs.launchpad.net/ironic/+bug/2063169

This is a project to create an simplified framework between Ironic and physical network configuration to facilitate orchustration of networking in a delineated way from existing OpenStack Neutron service in a model which would able to operated effectively by another team which is not a cloud team, but a network team.

The reasons why are plentiful:

- The number of Operators utilizing Ironic continue to grow, although the operators utilizing Ironic in fully integrated configurations is not growing at the same rate as operators running in a standalone mode.
- Operators needing physical switch management generally need to operate in an environment with strong enforcement of separation of duties. i.e. The software might not be granted access to the Switch management framework, nor can such a service be accessible by any users under any circumstances.
- The introduction of DPUs generally means that we now have potential cases where switches need to be programmed to provision a DPU, and then the DPU needs to be programmed to provision servers.

The goals can be summarized as:

- Provide a mechanism to configure L2 networks on Switches, which may be facilitated by a modified networking-generic-switch or similar plugin.
- Provide a mechanism to configure L2 networks to be provided to a host from a DPU.
- Provide a mechanism to accommodate highly isolated network management interfaces where operators restrict access such that *only* Ironic is able to connect to the remote endpoint.
- Provide a tool to apply and clean-up configuration, *not* track and then assert configuration. This doesnt preclude a future double check this configuration mode from existing at some point, but the minimal viable functionality is application and removal of the network configuration.
- Provide a mechanism of receiving the call to do something, reading in networking configuration credentials from local storage, and then doing so without the need of a database *OR* shared message bus.

This project is NOT:

- Intended to provide any sort of IPAM functionality.
- Intended to provide management of Routing.
- Intended to provide management of Security Groups or Firewalling.
- Intended to provide a public ReSTful API, nor require a database.

This project MAY:

- Provide a means to help enable and deploy advanced tooling to a DPU under Ironics control.

- Provide a means of offloading some of the layer2 interaction responsibility in an environment *with* Neutron and Ironic, espescially.

Warning

This document is not a precise and thus prescriptive design document, but an document to record and surface the ideas in a way that can foster communication and consensus building. In this case, we are likely to leave it to the implementers progotive with this document being overall guard rails.

Problem description

Today, Ironic has only one option to facilitate the automation of switch level infrastructure, which is to leverage Neutron and the ML2 interface. Unfortunately, infrastructure operators needing Ironic largely reject this model because the network is often owned by a separate group.

As a result we need a service to facilitate secure and delienated network management which can be owned and operated by separate infrastructure team in an enterprise organization, which brings together, aspects like simple code patterns and playbooks such that they can trust the interface layer to apply basic network configuration and enable easier use of Bare Metal. In most cases where this is needed, just the physical port needs to be set to a specific network; addressing is often separately managed and not our concern.

Furthermore, the available ecosystem in the DPU spaces wants to model their devices in a variery of ways and some of those devices have inherent limitations. For example, some devices are just another computer embedded and connected to the same PCI device bus. Others present the ability to load a P4 program to handle specific tasks. Or the available Flash and RAM of a device is highly limited such that options are very limited and entirely exclude all off the shelf operating systems and their utilities. This nature makes almost every device and their resulting use case entirely govern their configuration and use model which means an easy to modify modular interface would provide the greatest potential impact.

Proposed change

We are proposing an RPC service. Specifically something along the lines of a JSON-RPC endpoint, which multiple ironic conductors would be able to connect to in order to request networking changes to be made.

Along side of the RPC service, we would have an appropriately named `network_interface` driver to take the information stored inside of Ironic and perform attachment of interfaces based upon the provided information.

Note

A distinct possibility exists that we may actually start with a hybird dual `network_interface` driver to help us delineate and handle integration in a clear fashion. This is much more of an implementors progorative soft of item.

MVP would likely exclude locking, but be modeled as a single worker service or container which does not maintain state, largely simplifies the problem to who is logged into what to make concurrent changes, which has been the historical driver for locking.

Note

Teaching networking-baremetal to call this proposed RPC service is generally considered out of scope of this work, but entirely within reason and possibility to facilitate as this would provide functionally a capability for some calls to be proxied through and related to actions for a Neutron deployment to ultimately also call this new service.

The overall call model, at a high level could take the following flow.:

```
Inbound Request/Connection



{"type": "attach_port",
 "payload": {"context": {...}}}



Invoke Plugin With Context



Plugin handles locking, if necessary



Plugin succeeds (HTTP 200?) or fails and returns HTTP error
```

While originally envisioned to just be able to load an ML2 plugin directly, there plugins design model has some challenges in this context of remote execution, which is likely why a remote RPC model never evolved in Neutron.

Two basic issues:

1) Plugins, upon completing the binding action, update the database state in neutron through a pattern of updating the neutron database. This requires database access and credentials which are not available in our use case and model.

2) Plugins may also invoke methods on the original provided context. Context in Neutron, in this case, is not context provided by oslo.context, but an entirely separate construct consisting of past and future state information for the networking being modified.

So the obvious path forward is to simplify the model and design the RPC model used for this interaction around the basic actions, and allow for an ML2 user, the ability for the calls to be made which abstract the actions from the information/state/configuration updates.

- get_allowed_network_types - Returns a list of supported network types which can allow the caller to determine if the action can be supplied. Presently, baremetal engaging ML2 plugins largely just hard-code this as only supporting a VLAN type, but pushing this as far out as possible to a plugin being executed allows for it to be a generic pass-through.
- is_link_valid - Returns True/False depending on if the request has sufficient and correct information to be acted upon. Could be used for basic pre-action validation.

- is_port_supported - Returns True/False depending if the requested port is supported for actions. Generally this is a VNIC type check today, but could also be used by the remote service to perform basic pre-fliflight validation actions and allow a client to fail-fast.
- attach_port - Performs the actual action of attaching (Adding vlans or ultimately VNIs to a port).
- detach_port - Performs the actual action of detaching (removing network access from a port).
- update_port - Peroforms an attempt to update a port for attachment, such as if port-channels/bonding properties have changed.
- add_network - Adds a network to the remote device.
- delete_network - Deletes the network from the remote device.

In a sense, this changes the idea of just load an ml2 plugin to load either a hybridized ml2 plugin interface, OR just define our own plugin model which can be supported, and as such is an outstanding question this specification seeks to bring an answer to.

For the remote RPC service, it is anticipated that logging will need to be verbose enough that Operators can understand the questions they may raise when investigating issues. For example: When, Who, What, Why, and How. Plugin code in Ironic should **also** log verbosely when invoked to ensure operators can match requests and resulting changes should an issue arise.

While beyond an initial MVP of basic functionality, to solve the DPU case, the overall pattern model would likely take the shape of one where Ironic would enumerate through the child nodes, attach the child nodes to the requested physical network, and then engage on some level of programming which may need to be vendor or deployment specific based upon the overall use model. Details which at present time cannot be determined without the foundational layer needing to be constructed before being built upon.

From a users standpoint, the following sequence depicts their basic interaction and the overall resulting sequence.:

```
Existing node chosen by user



User posts to /v1/nodes/<ident>/vifs with payload
containing an id of a vlan ID.



User requests the node to deploy via the
/v1/nodes/<ident>/states/provision API interface



Ironic follows existing flow, triggering the new
network_interface module which calls this new
service to perform the attach/detach operations
in accordance with the existing model and node
lifecycle state.
Initial network in a deploy is the provisioning
```

(continues on next page)

(continued from previous page)

```
network.



Node deployment proceeds with resources already
connected to the desired network.



Once deployment has been completed, the network
interface module calls the new service to change
the attachment to the requested vlan ID. In the
event of a failure, the physical switch port
is detached.
```

Alternatives

The closest alternative would be a `standalone Neutron` coupled with some sort of extended/proxy RPC model, which is fine, but that really does not address the underlying challenge of the attach/detach functionality being needed by Infrastructure Operators. It also introduces modeling which might not be suitable for bulk infrastructure operators as they would need to think and operate a cloud model, as opposed to the physical infrasructure model. Plus operating Neutron would require a database to be managed, increasing operational complexity, and state would also need to still be navigated which increases the configuration and code complexity based upon different Neutron use models.

Another possibility would be to directly embed the network attach/detach loading and logic into Ironic itself, however that would present difficulties with maintenance where we largely want to unlock capability.

Data model impact

At this time, we are largely modeling the idea to leverage existing port data stored inside of Ironic which is utilized for attachment operations.

A distinct possibility exists we may look at storing some additional physical and logical networking detail inside of Ironics database to be included in requests, which could possibly be synchronized, but this would be beyond the scope of the minimum viable product as in the initial phase we intend to use the VIF attach/detach model to represent the logical network to be attached.

State Machine Impact

None

REST API impact

With a MVP, we do not anticipate any REST API changes to Ironic itself with the very minor exception of the losing of a Regular Expression around what Ironic accepts for VIF attachment requests. This was agreed upon by the Ironic community quite some time ago and just never performed.

Existing fields on a node and port will continue to be used just as they have before with an MVP.

Post-MVP may include some sort of /v1/physical_network endpoint to be designed, but that is anticipated to be designed once we know more.

Client (CLI) impact

openstack baremetal CLI

None

openstacksdk

None

RPC API impact

This change proposes a service which would be accessed by Ironic utilizing an RPC model of interaction. This means there would be some shared meaning for call interactions in the form of a library.

In all likelihood, this may be as simple as attach and detach, but given the overall needs of an MVP and a use model were focusing upon trying to leverage existing tooling as well, the exact details are best discovered through the development process which likely covers what was noted above in the Proposed Change section.

Driver API impact

None

Nova driver impact

None

Ramdisk impact

None

Security impact

Impact for Ironic itself is minimal, although it will require credentials to be set for the remote service to signal interface attachment/detachments.

The security risk largely revolves around the new service were looking at creating with this effort. The shared library utilized to connect to the remote service, likely needs to also contain the necessary client wrapper code, as an MVP service is likely going to start only with support for HTTP Digest Authentication, which can then move towards certificate authentication as it evolves.

In large part because that service will need to load and combine a set of credentials and access information. As such, this new service will **not** be a user facing service.

Today, individual ports are attached through a combination of network identifier and a binding profile which is utilized to map a port to a switch. In this model, there would be no substantial difference. A network_id would be a user supplied detail, and the local_link_information would contain sufficient information for the plugin executing to identify which device. The new service would retrieve the details to access the remote device from local configuration, and combine the rest of the binding profile and target network identifier to facilitate the attachment of the port to the device.

Note

This security impact does not denote the likely situation of DPU credential management. We are presently defferring the possibility as a challenge we would focus on after an initial minimum viable product state is reached.

Note

This security risk does not include any future mechanisms to do perform aspects such as software deployment on a DPU to facilitate a fully integrated with Neutron case, which is something we would want to identify and determine as we iterate along the path to support such a capability.

Other end user impact

None

Scalability impact

Please see the Performance Impact section below.

Performance Impact

This proposal is intentionally designed to be limited and isolated to minimize risk and reduce deployment complexity. It is also intentionally modeled as a tool to do something, and that something happens to be configuration in area where device locking is necessary. This realistically means that the only content written to disk is going to be lock files.

Furthermore, the possibility exists that the Ironic driver code utilized *could* wait for a response, where today Neutron port attachment/detachment calls are asynchronous. This would pose an overall improvement for end users of Ironic. This is solved today as a 15 second sleep by default, and might not be necessary in this design overall improving Ironic performance.

Other deployer impact

To utilize this functionality, deployers would need to deploy a new service.

This would be opt-in, and would not impact existing users.

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
<Volunteer #1>

Other contributors:
<Volunteer #2>

Work Items

Broadly speaking, the work items would include:

1) Prototyping this new service.
2) Wire up an ML2 driver such that we have interfaces we can load and call. This is anticipated to be neworking-generic-switch.
3) Prototyping an ironic network_interface driver to utilize this new service.
4) Test!

Note

The list below is intended to paint a picture of what we feel are the possible steps beyond the initial step of creating an Minimum Viable Product. They are included to provide a complete contextual picture to help the reader understand our mental model.

Past initial prototyping, the following may apply order:

- Creation of a common library for Ironic and any other program or tool to utilize to compose RPC calls to this service.
- Extend support to VXLAN ports, which may require additional details or design work to take place and work in any ML2 driver utilized.
- Design an API rest endpoint to facilitate the tracking of physical networks to be attached to baremetal nodes.
- Add support to networking-baremetal to try and reconcile these physical networks into Ironic, so node port attachment/detachments can take place.
- Add support to networking-baremetal for it to proxy the request through to this service for port binding requests in Neutron.
- Design a new model, likely superseding VIFS, but vifs could just also be an internal network ID moving forward. This would likely be required for formal adoption of the functionality by Metal3, but standalone users could move to leverage this immediately once implemented.
- Development of a model and flow where DPU devices could have a service deployed to them as part of a step invoked by Ironic. This would involve many challenges, but could be used to support the Neutron Integrated OVS/OVN agents to operate on the card for cases such as the remote card being in a hypervisor node.

Dependencies

To be determined.

Testing

An ideal model of testing in upstream CI has not been determined, and is dependent upon the state upon reaching a minimum viable product state, and then what the next objectives appear to be.

This may involve duplication of Ironics existing multinode job in a standalone form. Ultimately the expectation is we would have one or more CI jobs dedicated to supporting such functionality being exercised.

Upgrades and Backwards Compatibility

This functionality is anticipated to be net new for Ironic and exposed to end users through a dedicated `network_interface` module which could be selected by users at a point in the future. As such no upgrade or backwards compatibility issues are anticipated.

Documentation Impact

No impact is anticipated at this time.

References

https://etherpad.opendev.org/p/ironic-ptg-april-2024#L609

5.15.164 Merge Inspector into Ironic

https://storyboard.openstack.org/#!/story/2010275

This specification proposes merging the ironic-inspector project (*Inspector*) into the main Ironic project completely, making all its API feature a part of the Bare Metal API and deprecating the separate project.

Problem description

Inspector was born from a bunch of unmerged patches to Ironic back in the middle of year 2014 under the name *ironic-discoverd*. The Ironic team considered the patches, especially the idea of auto-discovery, going too much outside of the Bare Metal project scope. In early 2015, the Ironic team has agreed to move the already established and functioning project under its wing, changing the name to *ironic-inspector*. Over time, Inspector gained and then lost again its own core reviewer team, while becoming more and more aligned with the standard practices in the Ironic community.

Fast forward to year 2023, very few still standing members of the community remember this story. Why is Inspector separate from Ironic? is a common question among newcomers. There are few problems with the current state of things:

Maintenance load.
A great example is the SQLAlchemy 2.0 transition. Since *Inspector* has a separate database, we need to do the work twice.

Operating load.
One more service - one more problem. It has to be installed, secured and updated regularly.

Scaling/HA issues.
Ironic has been designed with horizontal scaling and high availability in mind; Inspector has not. To a certain extent, it makes sense: Inspector is only used occasionally. Still, in a really large deployment (like CERNs) even rarely used services are accessed often enough for the availability to matter.

The problem has been mostly mitigated by introducing support for a message bus (RabbitMQ) and membership tracking (ZooKeeper/etcd). Mostly - because the solution depends on two services that are normally present in OpenStack, but are not present in standalone solutions like Bifrost or Metal3[1].

Performance issues.
Since Inspector has a separate database, it has to maintain its own nodes table and synchronize it with Ironic periodically. This is inefficient and may contribute to scalability problems.

[1] Technically, any Kubernetes deployment includes etcd, but it is against best practices for non-Kubernetes applications to rely on it.

Similarly, every inspection hook (see the *Glossary*) has to issue some API requests to Ironic to update information. A pipeline with several hooks may generate noticeable traffic.

Resource utilization.
Inspector, being written in Python, has a non-negligible RAM footprint, while doing nearly nothing most of the time.

Discoverability.
Since Inspector has its own documentation, API reference and client, its harder to discover information about it.

Glossary

In-band inspection
Hardware inspection that is conducted through the use of a ramdisk with IPA (ironic-python-agent). The other option is out-of-band inspection, where the data is collected through the BMC without powering on the node.

The term *introspection* is used as a synonym in the Inspector code base and documentation.

Inspection data
Generic term for all information collected by IPA during inspection and sent back to Inspector (in the future - to Ironic).

Inventory
In the broad sense - synonym to inspection data. In the narrower sense - hardware inventory as defined by IPA and returned by its hardware managers.

Inspection collectors
IPA plugin responsible for collecting inspection data on the ramdisk side. The `default` one collects inventory. Collectors are currently independent from hardware managers, but this may change in the future.

Processing hooks
Inspector (in the future - Ironic) plugins for processing inspection data and updating the node as a result. Allow operators to fine-tune the inspection process by configuring a pipeline of processing hooks.

This document introduces the term *inspection* hooks to avoid ambiguity in the larger Ironic context.

Inspection hooks and collectors may correspond to each other. For example, the `extra-hardware` collector has a corresponding `extra_hardware` hook.

Managed/unmanaged inspection
We are talking about *managed* inspection when it is Ironic that completely sets up the boot environment, be it (i)PXE or virtual media boot. Inspection is *unmanaged* when the (i)PXE environment from Inspector, usually a separate *dnsmasq* installation, is used.

Unmanaged inspection has been the default for Inspector for a long time. Managed inspection was introduced to support virtual media, first and foremost. With the proposed merge, managed inspection will become the default mode, while unmanaged inspection will require an explicit configuration.

PXE filters
A way for the separate Inspectors PXE environment to co-exist with Neutron by limiting which MAC addresses it serves. Inspector has two implementations:

- `iptables` uses firewall to open/close access to a DHCP server

- `dnsmasq` updated *dnsmasq* host files with MAC addresses to serve/deny

Over time, the `dnsmasq` PXE filter was found to be more scalable and flexible (e.g. it supports spine/leaf architectures with DHCP relays) at the cost of supporting only one DHCP server implementation.

This specification proposes migrating only the `dnsmasq` PXE filter. The `iptables` one will follow if we see a demand for it (the deprecation release notes will mention it).

Auto-discovery
Process of enrolling new nodes automatically. When unmanaged inspection is configured, unknown nodes on the provisioning network will boot into the IPA ramdisk, get inspected and registered in Ironic.

This operation is disabled by default.

Proposed change

Inspector features will be migrated step-by-step into the Ironic code base, predominantly into the `ironic.drivers.modules.inspector` package. We will try to avoid radical changes in the design with the following exceptions:

- Split the poorly defined *inspection data* into the formally defined and version (although not in the sense of API microversions) inventory and free-form plugin data (influenced by inspection collectors in IPA and processing hooks in Inspector/Ironic - see the *glossary*).

  Inventory will not be mutated by any inspection hooks, thus reducing the need for having unprocessed inspection data API (which Inspector has).

- Avoid poorly documented internals data formats in the processed data. For example, Inspector generates fields `interfaces` and `all_interfaces` that are not based on the `interfaces` collection in the inventory.

- Split the migrated PXE filter into a new script to avoid coupling it (and thus the *dnsmasq* instance behind it) to Ironic processes. This way, it can be scaled separately.

- Rework inspection (former processing) hooks for more obvious naming and better composability.

  Consistently use dashes instead of underscores in entry point names.

  Fewer hooks will run in the default configuration.

- Consistently use the term *inspection* instead of *introspection*.

For the sake of keeping this specifications size reasonable and my sanity (relatively) intact, inspection rules are omitted here. Theyre relatively trivial, but require a lot of explanation and can be implemented independently.

Alternatives

Keep Inspector separate.
Possible arguments for it include:

- Better utilizing CPU cores by having a separate process. This should better be solved by allowing several conductors per physical host. Such a solution will benefit also more intensive operations and deployments without Inspector.

- More manageable (i.e. smaller) code base. However, a lot of code in Inspector exists only because its a separate project. This includes most of the database code, some of the API endpoints and the node synchronization routine.

Do not migrate some of the major features.
This will hinder the migration and will prevent us from ever deprecating Inspector.

Do not migrate PXE filters.
This will make auto-discovery in the presence of Neutron impossible. Auto-discovery is a commonly requested feature (example request).

Data model impact

Inventory table

Add a new table for storing inspection data when the Object Storage service is not available:

```
class NodeInventory(Base):
    """Represents an inventory of a baremetal node."""
    __tablename__ = 'node_inventory'
    __table_args__ = (
        Index('inventory_node_id_idx', 'node_id'),
        table_args())
    id = Column(Integer, primary_key=True)
    inventory_data = Column(db_types.JsonEncodedDict(mysql_as_long=True))
    plugin_data = Column(db_types.JsonEncodedDict(mysql_as_long=True))
    node_id = Column(Integer, ForeignKey('nodes.id'), nullable=True)
```

Here, `inventory_data` contains the inventory as defined by IPA (see the *Glossary*), while `plugin_data` contains auxiliary data returned by various collectors or generated by inspection hooks.

The `NodeInventory` object is deleted on node deletion.

Note

This table already exists at the time of writing this specification. It is included here for completeness.

Node modifications

Add a new boolean field `auto_discovered` to the `nodes` table. It will be read-only from the API standpoint and will be used to mark auto-discovered nodes.

State Machine Impact

The only expected change is a possibility of transition from *INSPECT WAIT* to *INSPECTING*, which is arguably missing by mistake.

REST API impact

Add a new API endpoint to fetch the inventory and the optional plugin data:

`GET /v1/nodes/{node}/inventory`
Returns a JSON object with two keys: `inventory` and `plugin_data`.

HTTP status codes:

- 200 on success.
- 404 if the node is not found, no inventory is recorded, or the API is not available in the requested version.

Note

This API already exists at the time of writing this specification. It is included here for completeness.

Add a new API to accept the inspection data from the ramdisk:

`POST /v1/continue_inspection`
Accepts inspection data in exactly the same format as Inspector, namely as a JSON object with at least an `inventory` field.

The only query parameter is `node_uuid` - an optional UUID of the node. This parameter is designed for virtual media deployments to safely pass the node identity to the ramdisk. See *lookup process* for some details.

This API is **not authenticated** and does not require an agent token since inspection always happens first on agent start-up.

The result depends on the requested API version:

- In the base API version, Inspector compatibility mode is used. The resulting JSON object has one key: `uuid` (the nodes UUID).
- In the new API version, the result is the same as in the normal lookup API. In this case, the API generates and returns an agent token, effectively replacing lookup when inspection is used.

HTTP status codes:

- 200 on success.
- 404 if the node is not found, several nodes match the provided data, or the node state is not `INSPECT WAIT`.

Note

We use the same generic HTTP 404 response to avoid disclosing any information to a potential intruder.

Return the `auto_discovered` field in the full node representation. Update the node listing (`GET /v1/nodes`) with an ability to filter by this field.

Lookup process

The lookup process is somewhat more complicated than the normal Ironic lookup because the inspected nodes may not have any ports enrolled. The procedure will try to find one and *only one* node that satisfies the provided node UUID, MAC addresses and BMC addresses.

BMC addresses are not indexed in the database and require some pre-processing. When *starting* inspection, BMC addresses will be collected from the nodes `driver_info`, resolved into IP addresses and

cached in the `driver_internal_info`. On lookup, `driver_internal_info` from all nodes in the `INSPECT WAIT` state will be checked.

If none or several nodes match the data, HTTP 404 with no explanation will be returned. Extensive logging will be provided for debugging purposes.

Auto-discovery

If auto-discovery is enabled (see *auto-discovery configuration*), the lookup process will work a bit differently for completely new nodes. If no node at all can be found for data (as opposed to a node in an invalid state), a new node will be created by the API layer. The rest of inspection happens the same way.

Nodes created this way will have an `auto_discovered` field set to `True`.

Client (CLI) impact

All new API features will need to be exposed.

openstack baremetal CLI

Starting and aborting inspection.
No changes here, use the same commands:

```
$ openstack baremetal node set --inspect-interface agent <node>
$ openstack baremetal node inspect <node>
$ openstack baremetal node abort <node>
```

Exposing inspection data.
I would like to have command to display certain parts of the inventory, filter it, etc. It is unclear if we should do it on the client or server side (or not do it at all). Inspector has a couple of comments for extracting parts of the inventory. I suggest not to migrate them in the first iteration.

Well start with migrating the most basic command, simply saving the complete JSON to a file or displaying it:

```
$ openstack baremetal node inventory save [--file <file path>] <node>
```

The callback endpoint will not be exposed via the CLI.

Filtering auto-discovered nodes
Can be useful for auditing purposes, especially together with filtering on provision state:

```
$ openstack baremetal node list --provision-state enroll --auto-discovered
```

openstacksdk

Expose a call to fetch inventory, very similar to the existing call for Inspector:

```
def get_inventory(self, node):
    """Get inventory for the node.

    :param node: The value can be the name or ID of a node or a
        :class:`~openstack.baremetal.v1.node.Node` instance.
```

(continues on next page)

(continued from previous page)

```
    :returns: inspection data from the most recent successful run.
    :rtype: dict
    """
```

Update the node API with filtering on `auto_discovered`.

RPC API impact

A new RPC call will be introduced for handling the inspection data:

```
def continue_inspection(self, context, node_id, inventory,
                        plugin_data=None):
    """Continue in-band inspection.

    :param context: request context.
    :param node_id: node ID or UUID.
    :param inventory: hardware inventory from the node.
    :param plugin_data: optional plugin-specific data.
    :raises: NodeLocked if node is locked by another conductor.
    :raises: NotFound if node is in invalid state.
    """
```

On receiving this call, the conductor will acquire an exclusive lock, double-check the provision state and launch a thread for further processing.

See *PXE filter script* for further impact.

Driver API impact

Extend the *inspect interface* with an additional call:

```
def continue_inspection(self, task, inventory, plugin_data=None):
    """Continue in-band hardware inspection.

    Should not be implemented for purely out-of-band implementations.

    :param task: a task from TaskManager.
    :param inventory: hardware inventory from the node.
    :param plugin_data: optional plugin-specific data.
    :raises: UnsupportedDriverExtension, if the method is not implemented
             by specific inspect interface.
    """
```

Inspection hooks

Inspection hooks are a new kind of Ironic plugins, closely based on the Inspectors processing hooks (see the *Glossary*).

The current Inspectors processing hook interface looks like this (shortening docstrings for readability):

```
class ProcessingHook(object, metaclass=abc.ABCMeta):

    dependencies = []
    """An ordered list of hooks that must be enabled before this one."""

    def before_processing(self, introspection_data, **kwargs):
        """Hook to run before any other data processing."""

    def before_update(self, introspection_data, node_info, **kwargs):
        """Hook to run before Ironic node update."""
```

Adapting to the Ironic terminology, new API and internal structures, this becomes:

```
class InspectionHook(metaclass=abc.ABCMeta):

    dependencies = []
    """An ordered list of hooks that must be enabled before this one."""

    def preprocess(self, task, inventory, plugin_data):
        """Hook to run before the main inspection data processing."""

    def __call__(self, task, inventory, plugin_data):
        """Hook to run to process the inspection data."""
```

Hooks

- **must** override `__call__` and *may* override the other two methods.
- are always run with an exclusive lock with the node in the `INSPECTING` provision state.
- *may* modify the plugin data but *should not* modify the inventory.
- *should avoid* permanently modifying the node or any related resources in the `preprocess` phase.
- **must** call `task.node.save()` explicitly on modifications.

The ordered list of hooks that will run by default (see *hooks configuration*):

`ramdisk-error`
: Fails the inspection early if an error message is passed along the inspection data.

`architecture`
: Sets the `cpu_arch` property based on the inventory.

`validate_interfaces`
: Validates interfaces in the inventory. Valid interfaces are stored in `plugin_data` in the new key `valid_interfaces` with an additional field `pxe_enabled`.

`ports`
: Creates ports based on the `add_ports`/`keep_ports` options (see *port creation configuration*). Requires the `validate_interfaces` hook. Updates the `valid_interfaces` collection with a new boolean interface field `is_added`.

The list of available optional hooks (adapted from existing Inspector hooks):

`accelerators`
: Sets the `accelerators` property based on the available accelerator devices and the configuration.

boot-mode
Sets the `boot_mode` capability based on the boot mode during the ramdisk run.

cpu-capabilities
Updates capabilities based on CPU flags from the inventory.

extra-hardware
Converts the extra collected data from the format of the hardware-detect tool (list of lists) to a nested dictionary. Removes the original `data` field from the `plugin_data` and creates a new field `extra` instead.

local-link-connection
Uses LLDP information to set the `local_link_connection` field on ports. Can be used together with `parse-lldp`.

memory
Sets the `memory_mb` property based on the inventory.

parse-lldp
Converts binary LLDP information into a readable form, which is then stored in the `plugin_data` as a new `parsed_lldp` dictionary with interface names as keys.

https://specs.openstack.org/openstack/ironic-inspector-specs/specs/lldp-reporting.html

pci-devices
Updates the nodes capabilities with PCI devices using a mapping from the configuration.

https://specs.openstack.org/openstack/ironic-inspector-specs/specs/generic-pci-resource.html

physical-network
Allows setting the ports `physical_network` field based on the CIDR mapping in the configuration.

Can be subclassed to implement a different logic.

raid-device
Uses a diff between two inspection to detect the freshly created RAID device and configure it as a root device.

> **Note**
>
> The current implementation caches devices in the nodes `extra`. We should rather fetch the old inventory for that.

root-device
Uses root device hints to determine the root device and sets the `local_gb` property.

> **Note**
>
> Nothing will set the `cpus` property. Its not used by Nova any more and should be removed from essential properties.

Nova driver impact

Fortunately, none.

Ramdisk impact

At the first pass, there will be no changes to the ramdisk. The new callback API will be fully compatible with its counterpart in Inspector.

The follow-up change will be to make lookup and inspection mutually exclusive: if inspection (at least its synchronous lookup part) succeeds, the token and node data are returned in the response, and the lookup is not needed.

Security impact

- This change introduces one more API endpoint without authentication. Knowing either UUID, MAC address or BMC address, an intruder can receive some information as well as the agent token (if it hasnt been retrieved yet) for a node in the `INSPECT WAIT` state.

  If in-band inspection is disabled or simply not used, no nodes will ever be in the `INSPECT WAIT` state since it is not used by out-of-band inspection implementations.

Other end user impact

None?

Scalability impact

The scalability impact on a deployment with in-band inspection will probably be net positive because periodic sync-ups of Inspector with Ironic will no longer be necessary.

Having PXE filters as a separate process means that they can be scaled separately from the rest of Ironic (e.g. it may make sense to keep them in an active/standby setup, while the rest of Ironic is active/active).

Performance Impact

The expected performance impact is also positive:

- Removal of the periodic task that synchronizes inspection results from Inspector to Ironic.
- More efficient database queries on inspection lookup and in PXE filters.

Storing inventory in the database does impact its size, but it is already the case for Inspector. However, during the transition period, there will be two copies of inventory. If this becomes a problem, an operator may opt to disable the inventory storage on the Ironic side until ready to switch over completely.

Other deployer impact

Hooks configuration

New configuration options in the `[inspector]` section:

`default_hooks`
: A comma-separated lists of inspection hooks that are run by default. In most cases, the operators will not modify this.

The default (somewhat conservative) hooks set will create ports and set `cpu_arch`.

hooks
A comma-separated lists of inspection hooks to run. Defaults to `$default_hooks`.

Note

This scheme allows easily inserting hooks in the beginning or the end of the list without hardcoding the default list, e.g.:

```
[inspector]
hooks = my-early-hook,$default_hooks,later-hook-1,later-hook-2
```

Port creation configuration

Various inspection hooks will come with their configuration. The most important is the port creation options in the `[inspector]` section:

add_ports
Which interfaces to enroll as ports for the node. Options:

- `all` (the default) - all valid interfaces.
- `active` - only interfaces with an IP address.
- `pxe` - only the PXE booting interface.

keep_ports
Which existing ports to keep.

- `all` (the default) - keep all ports, do not delete anything.
- `present` - delete all ports that do not correspond to interfaces in the inventory.
- `added` - delete all ports except for ones selected via the `add_ports` option (only makes sense if `add_ports` is not set to `all`).

Disk spacing configuration

An odd quirk of our partitioning code is that the `local_gb` field has to be smaller than the actual disk, otherwise the partitioning may fail. Inspector has been dealing it by making `local_gb` 1G smaller. This will be reflected in the following option:

disk_partitioning_spacing
Size in GiB to leave reserved. Defaults to 1, set to 0 to disable.

Auto-discovery configuration

The new `[auto_discovery]` section will have these options:

enabled
Boolean field, defaults to `False`.

driver
The driver to use for newly enrolled nodes. Required when the feature is enabled.

Note

Inspector has several options to tune the freshly created nodes. I believe that this complex logic should rather be implemented with inspection rules. The follow-up inspection rules spec will have some additions to make it easier.

PXE filter script

A new executable `ironic-pxe-filter` will be introduced to support unmanaged inspection in environments with Neutron. It will be designed to be deployed alongside a separate *dnsmasq* process. Unlike the current implementation in Inspector, the script will have direct access to the Ironic database for efficiency.

Operators that do not need PXE filtering, e.g. because they only use managed inspection or use a single PXE environment (without Neutron), can opt out of running `ironic-pxe-filter`. This applies, for example, to Bifrost and Metal3.

The only downside of this approach is knowing when inspection starts or finishes. Inspector learns it immediately and is able to update the filters without a further delay. The periodic task approach will result in a delay that is probably acceptable for real hardware but will be problematic for virtual machines.

To overcome this limitation, the new executable will feature an RPC service when the RPC transport is *oslo.messaging*. This service will use a separate *topic* `ironic.pxe_filter` and will receive broadcast messages from the conductor handling inspection. The RPC call will be:

```
def update_pxe_filters(self, context, allow=None, deny=None):
    """Update the PXE filter with the given addresses.

    Modifies the allowlist and the denylist with the given addresses.
    The state of addresses that are not mentioned does not change.

    :param allow: MAC addresses to enable in the filters.
    :param deny: MAC addresses to disable in the filters.
    """
```

No notifications will be done for JSON RPC transport. This will be documented as a known limitation. The further work as part of the cross-conductor RPC effort may eventually lift it.

Developer impact

While other in-band inspection implementations are possible, theyll probably happen as downstream modifications to the proposed implementation.

Implementation

Assignee(s)

Primary assignee:
Dmitry Tantsur (IRC: dtantsur, dtantsur@protonmail.com)

Other contributors:
Jakub Jelínek (IRC: kubajj) - inventory API

Work Items

Too many to mention - see tasks in https://storyboard.openstack.org/#!/story/2010275.

Dependencies

None so far.

Testing

- Bifrost will be migrated to the new implementation as early as possible.
- DevStack CI coverage will be added, possibly in form of tests in standalone jobs.
- Eventually, the existing Inspector job will be migrated over or deleted.

Upgrades and Backwards Compatibility

Other than the eventual deprecation of Inspector itself and the corresponding *inspect interface*, the change is backward compatible on the Ironic side.

Migration will be reasonably easy, but not necessarily friction-free. Possible concerns:

- No migration for inspection data. We could provide a tool, Im just not sure if its worth the effort. Can be done as an afterthought.
- Co-existence of the new and old *inspect interfaces*.
  - The callback API will be designed to work with the old interface by proxying the data to Inspector. This way, an operator can use the same callback URL for both implementations.
  - The new PXE filters script will also function the same way for both implementations.
  - The inventory API will be implemented for the old implementation by fetching the data from Inspector on successful inspection.

Documentation Impact

A lot of documentation has to be written, or rather adapted from ironic-inspector. API reference will be added for all new API endpoints.

References

5.15.165 Hardware that cannot be powered off

https://bugs.launchpad.net/ironic/+bug/2077432

Problem description

Power off is a very fundamental action for bare-metal provisioning. Not only is it available as an API primitive, Ironic also often uses a sequence of power off and power on instead of a single reboot action. This happens for three reasons:

- By waiting for the machine to power off first, we ensure that the power off request actually came through. Some IPMI implementation are notorious for ignoring power requests under certain conditions.
- Some actions require the machine to be off to work correctly or at all. For example, some hardware was reported to refuse to mount virtual media devices on a powered on machine.

- When multi-tenant networking is used, its essentially to switch the networking with the machine powered off, otherwise the code running on it will be exposed to both networks (e.g. IPA will stay running on the tenant network).

Unfortunately, in some cases powering off a machine is not possible:

- Some implementations of the NC-SI technology suffer from a serious drawback: the NIC that is shared with the BMC is powered off whenever the machine is powered off. When that happens, it is no longer possible to power on the machine remotely.
- DPUs may not support the power off action at all, relying on the parent machine power state instead. They may support reboot/reset though.

While the second case is related to an emerging technology, the first case is already seen in the wild and causes issues with the adoption of Ironic.

Proposed change

Add a new optional node flag `disable_power_off`. When set to `True`, Ironic will avoid ever issuing an explicit power off request. Specifically:

- All *power interfaces* will use an explicit reboot request (or fail if its not available) even when normally they would use a power-off/power-on pair (e.g. `redfish`). To detect the reboot, well insert a hardcoded sleep and then wait for the machine to be on.
- The `tear_down_agent` deploy step will no longer try to power off the machine. Instead, after collecting the ramdisk logs, it will issue the new `lockdown` IPA command to disable IPA (see *Ramdisk impact*).
- The `boot_instance` deploy step will unconditionally use hard out-of-band reset instead of the in-band power off command. The previously issued `lockdown` command will ensure that the disk caches are flushed already.
- The `tear_down_inband_cleaning` function will issue a reboot request after de-configuring IPA via `clean_up_ramdisk`. Same for `tear_down_inband_service`.
- On deployment, cleaning, inspection or servicing failure, the machine will stay on with IPA running.
- Validation will fail for any requested deploy, clean or service steps that include an explicit power off command.

Downsides

- The usage of `disable_power_off` opens up a potential vulnerability in the multi-tenant networking because IPA will be available on the tenant network for a short time. To mitigate this problem, a new *lockdown mode* will be introduced to IPA to ensure its at least not operational any more - see *Ramdisk impact*.
- Similarly, during cleaning the instance operating system will be switched to the cleaning network before IPA boots. Well document this as a potential issue and add a new configuration option to enable the no-power-off mode together with the `neutron` interface.
- After `tear_down_agent`, the machine will still be running. Any custom deploy steps or out-of-band actions that rely on the machine to be powered off after this step may fail.
- In case of IPMI, if the BMC ignores the reboot request, well still mark it as successful.

Alternatives

Short of convincing the vendors to fix the NC-SI issue in hardware, I do not see any alternatives. The NC-SI setup seems to be gaining popularity, especially in *far edge* setups.

The first version of this specification suggested adding `disable_power_off` to `driver_info` instead of making it a first-class node field. I changed it for two reasons: because of how many unrelated places in Ironic will need to check this field and to provide an easy way to guard access to it via RBAC.

Data model impact

None

State Machine Impact

None

REST API impact

A new field will be possible to set on node creation and later on. The access to it will be guarded by a new microversion and a new RBAC rule.

Client (CLI) impact

openstack baremetal CLI

Expose the new field as `--disable-power-off` to the create/set commands.

openstacksdk

Expose the new field on the Node object.

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

Add a new command `lockdown` that will prepare the machine for a hard reset and make sure IPA is not practically usable on a running ramdisk. Namely:

- Issue `sync` and write 3 to `/proc/sys/vm/drop_caches` to flush Linux caches.
- For each device, issue `blockdev --flushbufs <device>` to flush any outstanding I/O operations.

- Stop the heartbeater thread and the API.
- Try to disable networking by running `ip link set <interface> down` for each network interface.

Security impact

See *Downsides* for security trade-offs that need to be made.

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

A new security parameter will be added:

`[neutron]allow_disabling_power_off` (boolean, default `False`)
If `False`, the validation of the `neutron` network interface will fail for nodes that have `disable_power_off` enabled. If set to `True`, this feature will be usable together.

Developer impact

Authors of 3rd party power interfaces must take the new flag into account. Well give them a heads-up via release notes.

Implementation

Assignee(s)

Primary assignee:
Dmitry Tantsur (dtantsur)

Other contributors:
TBD

Work Items

- Update all power interfaces to respect the new flag.
- Update the agent deploy steps to respect the new flag.

Dependencies

None

Testing

Its possible to add a standalone job that tests the new mode of operation. We could even modify sushy-tools to reject power-off calls, but using this approach in the CI would require a new job, and were trying to avoid new jobs.

Upgrades and Backwards Compatibility

No concerns

Documentation Impact

Add a documentation page that lists the use cases and highlights the drawbacks.

References

5.15.166 Pluggable network providers

https://bugs.launchpad.net/ironic/+bug/1526401

Today, Ironic provisions servers on the same (flat) network that tenants run on. Ironic should have the ability to logically separate provisioning and tenant networks, and switch between them as needed.

Note: where this spec says network, it generally means Neutron network, which may be implemented in different ways.

Problem description

Ironic currently provisions servers on the same (flat) network that tenants use. It is clear that Ironic should allow operators to separate provisioning and tenant networks; where deploy ramdisks run on the provisioning network, and instances run on one or more tenant networks.

In order to secure the provisioning network, Ironic should be able to switch between these networks when spawning and destroying instances.

This method should also be extended to other management networks that may or may not be separated from the provisioning network; for example the operator may wish to use different management networks for cleaning or rescue tasks.

Proposed change

Ironic should have a pluggable network provider that can handle switching nodes between different networks. This provider should be able to be selected per Ironic node object. A new *network_provider* field will be added to the node object to define which network provider to use for that node. There should also be a configuration option for the default network provider, defaulting to none for now. The default value for *node.network_provider* will be NULL, meaning use the configuration option. Both the node field and the configuration option may be set to none or neutron, mapping to the no-op provider and the Neutron provider, respectively.

This network provider system will not be part of Ironics driver interface mixin system; rather it is standalone and is loaded on demand via stevedore. This is similar to how the DHCP provider code works today.

An initial implementation of a no-op network provider should be written, to put the logic in place in Ironic while maintaining compatibility with the existing model.

A second implementation should be written that uses Neutron to attach hardware to networks as needed, and should work as follows.

The network provider should have a concept of a provisioning network, where the control plane can communicate with nodes managed by Ironic in order to deploy, tear down, or otherwise manage nodes. It should be able to connect and disconnect nodes from this network.

The network provider should also know how to connect and disconnect nodes from arbitrary tenant networks. These networks are defined by the Nova user and Nova. Nova should continue to create Neutron ports (a logical attachment to a network), but these ports will be left unbound, as they will not have enough information to be plumbed through. Ironic later should send a port-update call to Neutron, to pass the necessary information to complete the binding. This call should happen after deploying the image, between the power off and power on calls that boot to the instance image. This may have implications for boot from volume workloads. As Ironic does not yet support these workloads, these are out of scope for the purposes of this spec.

In the case where the Ironic driver is used, Nova should send a null `host_id` in the binding profile. This will prevent Neutron from binding the port immediately, so we can defer this and allow Ironic to tell Neutron to bind the port when it is ready to do so. Ironic should send the node UUID as the `host_id`. Ironic will also delete the Neutron port connecting the node to the provisioning network at this time. The reverse will happen at tear down time.

If an older client (e.g. Nova) is in use and does initially send the `host_id`, Ironic needs to handle this. There are two cases here:

- The node is using the Neutron network provider. In this case, Ironic should fetch the ports first, and if the ports are already bound with the correct info, do nothing. If binding failed due to missing switchport information, Ironic should update the port appropriately and allow it to be bound.
- The node is using the none network provider. In this case, the node is expected to be on the provisioning network after deployment (todays current model). In this case, the ports should be treated as they are today, putting DHCP configs on those ports, etc.

Nova and Ironic should both use the binding:profile dictionary to send data such as physical switchport information.

Nova currently has a broken assumption that each Ironic port (physical NIC) may only attach to one network. This assumption will need to be fixed, as hardware (today) cannot spawn arbitrary NICs like virtual servers can. We may, in the future, also need a way for Nova users to define which NIC is attached to which networks. A first version should leave this assumption in place for the sake of simplicity.

If port groups exist for a node, those should be connected to the networks rather than the individual port. This allows for an aggregated connection such as a LAG to connect to the network.

Note that an Ironic environment may be deployed without Nova. In this case, the user should make the same calls Nova would make.

One caveat here is that nodes will not be able to PXE boot the instance image if they cannot reach the conductor (for tftp). Local boot will need to be used for any node deployed outside of the provisioning network. Any deploys outside of the provisioning network that do not use local boot should error.

Deploy drivers should call to this interface at proper times to switch between networks. For example, a driver completing a deploy should power off the node, switch to the instance networks, and power on the node. This will ensure that the deploy ramdisk never runs on the instance network, and the instance image never runs on the provisioning network.

Alternatives

Alternatively, Ironic could continue to prescribe that operators run Ironic on a single flat network shared between tenants and the control plane. This is clearly not viable for many real-world use cases.

Data model impact

A *network_provider* field will be added to the Node object.

State Machine Impact

None.

REST API impact

Update the REST API for the node object to allow reading and modifying the new network_provider field. This will likely need a version bump.

Client (CLI) impact

Will need to update the CLI to print the new Node.network_provider field, when available.

RPC API impact

None.

Driver API impact

This adds a new interface, *NetworkProvider*. This interface is *not* a part of Ironics driver composition system, to be clear. This interface will define the following methods:

```
def add_provisioning_network(self, task):
"""Add the provisioning network to a node."""

def remove_provisioning_network(self, task):
"""Remove the provisioning network from a node."""

def add_cleaning_network(self, task):
"""Add the cleaning network to a node."""

def remove_cleaning_network(self, task):
"""Remove the cleaning network from a node."""

def configure_tenant_networks(self, task):
"""Configure tenant networks (added by Nova/user) for a node."""

def unconfigure_tenant_networks(self, task):
"""Unconfigure tenant networks (to be removed by Nova/user) for a node."""
```

Nova driver impact

The Nova driver should not be directly impacted here; however, this does depend on changes to the Neutron network driver in Nova as described above.

Ramdisk impact

N/A

Security impact

This potentially improves security by restricting tenant access to the control plane.

Other end user impact

To use this feature, end users will need to:

- Set nodes to use the Neutron provider.
- Use local boot for nodes using the Neutron provider.

Scalability impact

When configured to use the Neutron plugin, this will result in additional API calls to Neutron to manage a node. However, impact on scalability should be negligible.

Performance Impact

None.

Other deployer impact

Two new configuration options will be added:

- `CONF.provisioning_network` specifies the ID of the provisioning network.
- `CONF.default_network_provider` specifies the default network provider to use for nodes with *node.network_provider* set to NULL.

A new database column (Node.network_provider) is also added, and so deploying this change will require a database migration to be ran.

Deployers will need to deploy a version of Nova that supports this feature, if using Nova.

Deployers will need to deploy an ML2 mechanism driver that supports connecting baremetal resources to Neutron networks.

Developer impact

Driver authors should support this feature by calling the methods provided.

Implementation

Assignee(s)

jroll <jim@jimrollenhagen.com>

And hopefully many others! :)

Work Items

- Add the Node.network_provider field and the default_network_provider configuration option..
- Implement the base interface.
- Implement the no-op provider.
- Instrument each deploy driver with calls to this interface.
- Implement the Neutron plugin provider.
- Modify Nova to send the extra flag discussed above, when creating ports for a machine using the Ironic virt driver.

Dependencies

None.

Testing

The no-op provider will be tested in the gate by default.

Neutron will provide an ML2 mechanism that simulates connecting real hardware to real switches. When that mechanism is available, we can test the Neutron provider in the gate.

Upgrades and Backwards Compatibility

Default behavior is the current behavior, so this change should be fully backwards compatible.

Documentation Impact

This feature will be fully documented.

References

Discussions on the topic include:

- https://etherpad.openstack.org/p/YVR-neutron-ironic
- https://etherpad.openstack.org/p/liberty-ironic-network-isolation
- Logs from https://wiki.openstack.org/wiki/Meetings/Ironic-neutron
- The spec for the rest of the API and data model changes, and ML2 integration in general: https://review.opendev.org/#/c/188528

5.15.167 New driver in Ironic for OneView

https://bugs.launchpad.net/ironic/+bug/1526406

This spec proposes adding a new driver that supports deployment of servers managed by OneView. OneView is an Integrated Infrastructure Systems Management Software developed by HP.

In this spec, *Server Hardware* is the label used on OneView to denote a physical server.

Problem description

Currently Ironic does not have integration with any Infrastructure Management System. Nowadays, being able to use hardware from these systems inventory to provision a baremetal instance is a manual/time consuming task that would require pre-configuration of each server. OneView eases this configuration workload.

This spec proposes a new Ironic OneView driver that will promote integration with the HP OneView Management System. The proposed driver will provide automatic inventory management with OneView, allowing Ironic to borrow non-dedicated servers from OneViewss inventory to provision baremetal instances with minimal common pre-configuration, set through OneViews *Server Profile Templates* (SPT).

In order to use a Server Hardware managed by OneView, one needs to assign it a *Server Profile*, prior to the enrollment of the node, in order to configure the hardware to be used (select/update firmware, enable NICs, setup the network connections on them, BIOS/UEFI settings, initialize storage and so on). The cloud administrator can take advantage of such a resource to configure the servers NIC to use the Ironic provision network on the fly as needed, along with other options that could allow optimal performance.

Proposed change

This spec proposes the *pxe_oneview* and *agent_oneview* drivers, implementing the Power, Management and, in the case of the Agent driver, Vendor interfaces.

The driver uses *python-oneviewclient* in order to handle the communication between the driver and OneView for, e.g., getting information about a resource, turning a server hardware on and off, and handling the configuration of a server profile.

Server profiles are based on SPTs, which have information to configure the hardware NIC to join a flat network, initialize the server storage and other configuration options used by Ironic like boot_type and boot_order. The SPT can also hold specific configuration options to improve the hardware performance for Ironic, like Advanced Memory Protection, USB boot, enable Virtualization Technology and Hyper-Threading, change the thermal configuration and so on. Based on this premises, to be enrolled, the node MUST have the following parameters:

- **driver_info**
  - server_hardware_uri: URI of the Server Hardware on OneView
  - server_profile_template_uri: URI for the Server Profile Template used to create the Server Profile of the node. This will be used on the future to change the Server Profile of the node on a zapping task.
- **properties/capabilities**
  - server_hardware_type_uri: URI for the Server Hardware Type on OneView, for scheduling purposes if one wants to deploy on specific hardware determined on the flavor.
  - enclosure_group_uri: URI for the Enclosure Group on OneView, for scheduling purposes if one wants to deploy on specific enclosure determined on the flavor.

The driver implements:

- oneview.power.OneViewPower
- oneview.management.OneViewManagement
- oneview.vendor.AgentVendorInterface

Power Interface:
The *_oneview drivers Power Interface controls and synchronizes the power state of the nodes using OneViews REST API. The validate() method on this interface will check the required parameters and if the node already has a server profile associated.

Management Interface:
The *_oneview drivers Management Interface allows the user to get and set the boot-order of a server hardware by modifying the server profile assigned to the server hardware. If no server profile is assigned yet to an instance, an exception will be thrown since the boot order of a server managed by OneView can only be modified through a server profile. The validate() method on this interface will also check the required parameters and if the node already has a server profile associated.

Agent Vendor Interface:
The *agent_oneview* interface modifies the way *reboot_to_instance* method sets the boot device since OneView doesnt allow such a change with the machine powered on.

This driver reuses PXEBoot for boot and ISCSIDeploy/AgentDeploy for deploy.

To be deployed using the *_oneview driver, the nodes Server Profile MUST be applied to the server hardware the node represents. This Server Profile MUST connect the 1st NIC of the node to Ironics provision network.

Alternatives

We could use the already existing drivers (such as pxe_ipmitool, pxe_ilo, iscsi_ilo or even agent_ilo) to launch instances managed by OneView. But then: - We would lose the capability to manage these instances through OneView; - If the node is being managed by OneView, without a Server Profile, and deployed with other driver, another user can claim it by applying a Server Profile and thus Ironic would lose control of the server; - Without using OneView, the task of maintaining configuration consistency between the Server Hardware items is manual, boring and time consuming.

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

N/A

Security impact

The connection with OneView is by default secure using TLS with certificate authentication, but the user can allow insecure connections by setting to True the allow_insecure_connections field in the configuration file.

Other end user impact

None

Scalability impact

The driver gets some data using *python-oneviewclient* through OneViews REST API which is an external service. The calls are simple, but considering a large amount of Server Hardware items a small increase in network traffic can happen.

Performance Impact

None

Other deployer impact

The following parameters are required in the newly created [oneview] section on ironic.conf:

- manager_url: OneView Manager url
- username: User account with admin/server-profile access privilege in OneView
- password: User account password in OneView
- allow_insecure_connections: Allow connections to OneView without a certificate signed by a trusted CA. Its default value is False.
- tls_cacert_file: The path to the certificate of a trusted CA to be used to verify the OneView certificate when insecure connections are not allowed

- max_polling_attempts: Max connection attempts to check changes on OneView

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
thiagop

Other contributors:
albertoffb caiobo diegolp liliars sinval afaranha

Work Items

- Implement new *iscsi_pxe_oneview* and *agent_pxe_oneview* drivers.
- Implement unit-test cases for **_oneview* driver.
- Write configuration documents.

Dependencies

- The driver requires python-oneviewclient package.

Testing

Unit-tests will be implemented for the new drivers. A third party CI will be used in the future to provide a suitable test environment for tests involving an OneView appliance.

Upgrades and Backwards Compatibility

None

Documentation Impact

The required parameters on the node and *[oneview]* section of *ironic.conf* will be included in the documentation to instruct operators how to use Ironic with OneView.

References

OneView Page
http://www8.hp.com/ie/en/business-solutions/converged-systems/oneview.html

OneView REST API Reference
http://h17007.www1.hp.com/docs/enterprise/servers/oneviewhelp/oneviewRESTAPI/content/images/api/index.html

python-oneviewclient
https://pypi.org/project/python-oneviewclient

5.15.168 New Release Model for Ironic

This specification describes the new model of releasing deliverables under the ironic umbrella, including the Bare Metal service itself.

Problem description

Currently ironic follows the OpenStack release model *cycle-with-intermediary* that permits us to produce one or more releases per OpenStack cycle, with the last one in a cycle used as a final release as part of the OpenStack integrated release (called *named release* in this document, as it receives a code name). A stable branch is created from this final release for long-term bug fix support.

As the ironic community is exploring an increasing number of standalone applications, including ones outside of OpenStack (such as Metal3), one problem has become apparent: the stand-alone usage requires not just frequent feature releases, but rather **supported** frequent feature releases. To be precise, we would like to produce supported releases every 1-3 months (2 months is what this spec proposes), where *supported* involves:

- Addressing at least critical issues with a point release. Currently we require consumers to switch to the next release, even if it is major.
- A supported upgrade path between releases. Currently only upgrades between OpenStack named releases are tested and supported, we need to support upgrades between intermediate releases as well.
- Documentation for each release. Currently only documentation for named releases (and master) is published.
- An obvious way to consume such releases. Currently deployment tools, including even Bifrost, are oriented on named releases or git master.

The proposed model aims to implement these requirements while keeping the existing commitments around the integrated release.

Proposed change

Terminology

The following concepts are used throughout this document:

named release
: is an integrated OpenStack release that receives a code name and a *named stable branch*.

intermediate release
: is a release of an ironic deliverable that happens as part of the master development (purposely excluding stable releases here) that does not correspond to a named release.

> **Note**
>
> This definition differs from the official OpenStack definition of an intermediary release. This is done on purpose to make the wording of this document clearer.

stable branch
: a branch created from a named release where bug fixes are applied and periodically released as *stable releases*.

stable release
is a release created as part of stable support of a *named* release.

bugfix branch
a branch created from an intermediate release that did NOT result in a stable branch.

Releasing

- Releases for all deliverables are evaluated on a loose bi-monthly basis, i.e. roughly every 2 months. If there are significant valuable changes, and a clear user interest in performing a release, we should do so. In most cases, this means a downstream distributor of Ironic is going to consume the release.

  This gives up to 6 releases a year, 3 per each OpenStack cycle.

- Two releases a year correspond to OpenStack named releases, the others happen between named releases. The former two happens always, the latter 4 can be skipped if there are no known consumers for that release or a deliverable does not see notable changes within 2 months.

- One week of soft feature freeze is observed before every release. *Soft* implies that feature can still merge if they are considered low-risk or critical for the scope of the upcoming release.

  This leaves merge windows of roughly 7-9 weeks for most features. Plans for them should be made during the feature freeze of the previous release.

- Ironic deliverables follow SemVer with one clarification: *patch* releases are always issued from a stable or bugfix branch (see *Stable branches and upgrades*). Releases from master always receive a minor or major version bump.

  Note

  This limitation is required to be able to find a suitable version for a branch release. E.g. imagine we cut 21.2.0 from master, then 21.2.1 also from master. If then we need to make a release from `stable/21.2` (the support branch for 21.2.0), there is no SemVer version to use (21.2.0.1 is still an option, of course, but may conflict with pbr).

- Intermediary (non-named) releases will target standalone users. OpenStack deployers will be recommended to use named releases.

Stable branches and upgrades

Service projects and bifrost

The following procedure will be applied to service projects (ironic and ironic-inspector), ironic-python-agent and bifrost:

- A stable branch is created from each release:
  - A traditional `stable/<code name>` branch for releases that coincide with named ones.
  - A `bugfix/<major.minor>` branch for other releases.

  The naming difference highlights the difference of intents:
  - *Stable* branches are designed for long-term consumption by downstreams (such as RDO) and for users to follow them.

  - *Bugfix* branches are a technical measure to allow patch releases after a certain release. Users and downstreams are not expected to follow them over a long period of time and are recommended to update to the next feature release as soon as it is out.

- Three upgrade paths are supported and tested in the CI for each commit:
  1. Between two subsequent named releases (e.g. *Train* to *Ussuri*).
  2. Between two subsequent intermediate releases.

     Note

     Its unlikely that well be able to use Grenade for that. Well probably use Bifrost instead.

  3. From a named release to any intermediate release in the next release cycle.

     Note

     Supporting this path is technically required to implement CI for the other two paths).

Note

Operating CI on the non-named branches may require pinning devstack, tempest and ironic-tempest-plugin versions to avoid breakages. It will be determined on the case-by-case basis.

Other projects

Library projects (metalsmith, sushy, python-ironicclient and python-ironic-inspector-client) and networking plugins (networking-baremetal and networking-generic-switch) will be released and branched as before:

- Releases will be created on demand based on how many useful changes are available.
- Only named stable branches will be created, intermediate releases will not result in branching.

This procedure matches how libraries are usually released in the Python world.

The CI tools (virtualbmc and sushy-tools) and ironic-tempest-plugin will not be branched.

Support phases

- A named stable branch is supported according to the OpenStack policies, which is currently 1.5 years of full support followed by extended maintenance.
- Since this proposal significantly increases the number of branches in support, well tighten the rules around backports to named branches:
  - The first 12 months any bug fixes are acceptable. Low-risk features **may** be accepted if theyre believed to substantially improve the operator or user experience.
  - The last 6 months and during the extended maintenance phase only high and critical bug fixes are accepted.

  - If during the extended maintenance no changes merge to a branch within 6 months, this branch is considered abandoned and is closed for further backports.

    Note

    This also applies when the changes are proposed but cannot merge because of failing CI.

- Bugfix branches (for deliverables that have them) are supported for 6 months. Only high and critical bug fixes are accepted during the whole support time.

  Note

  It may mean that a stable branch created earlier will receive more fixes than a bugfix branch created later. This is a reflection of the fact that consumers are not expected to follow bugfix branches.

- As before, high and critical bug fixes **should** be backported to all supported branches once merged to master.

Dependencies

Dependencies handling for named releases and branches does not change. For example, we keep consuming upper-constraints of a corresponding branch.

For intermediate releases we will consume upper-constraints from a future named branch. E.g. for Victoria we would consume https://releases.openstack.org/constraints/upper/victoria.

The inter-service dependencies for both named and intermediate releases must be expressed separately, both via microversioning or via documentation. We already provide support for a broad set of versions of projects we can integrate with.

Deprecation policy

The deprecation policy remains intact: any deprecated functionality can only be removed after 6 months pass and a **named** release is done.

Alternatives

- Keep the current model, ask intermediate releases consumers to always upgrade to the latest one.

Data model impact

None

State Machine Impact

None

REST API impact

None

Microversioning is already used as a way to ensure cross-releases API compatibility.

Client (CLI) impact

None

openstack baremetal CLI

None

openstacksdk

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

We expect the Nova driver released as part of a certain OpenStack release series to be compatible *at least* with all Ironic releases from the same series and with the last release from the previous series.

Ramdisk impact

Under the proposed model, ironic, ironic-inspector and ironic-python-agent will get released at roughly the same time. The compatibility rules will be:

Each release of ironic/ironic-inspector is compatible with

- the release of ironic-python-agent that happens at the same time
- the last named release of ironic-python-agent

Note

Supporting releases between these two is very likely but is not officially guaranteed nor tested in the CI.

Each release of ironic-python-agent is compatible with

- the releases of ironic and ironic-inspector that happen at the same time
- the last named releases of ironic and ironic-inspector

Note

The first 3 rules are already enforced in the CI, the last will require a new job on ironic-python-agent, supposedly based on Bifrost.

The compatibility matrix will be provided through the documentation as part of the pre-release documentation update and via the future web site.

We will publish ironic-python-agent images corresponding to all stable branches, named and intermediate (currently images are only published for named branches) and provide instructions on how to build customized images based on a certain branch or release.

Security impact

Supported intermediate releases will also receive security bug fixes.

Other end user impact

See *Other deployer impact*.

Scalability impact

None

Performance Impact

None

Other deployer impact

Deployers will have faster access to new features if they opt for using intermediate releases.

Developer impact

No direct impact. The *Deprecation policy* is not changed.

Implementation

Assignee(s)

The whole team is expected to be responsible for executing this plan, the primary assignee(s) will coordinate it.

Primary assignee:
Dmitry Tantsur (@dtantsur, dtantsur@protonmail.com)

Work Items

- Discuss this document with the release team and the TC. Make necessary adjustments to our deliverables in the release repository.
- Update the releasing documentation and publish our release schedule.
- Create new CI jobs as described in *Testing*.

- Start publishing ironic-python-agent images from non-named stable branches (may work out-of-box).
- Update Bifrost to support installing components from latest published releases.

Dependencies

None

Testing

Two new family of the CI jobs will be introduced:

- Intermediary upgrade jobs on ironic and ironic-inspector, testing upgrade from the last intermediate release branch.
- Backwards compatibility job on ironic-python-agent to test every commit against the previous named releases of ironic and ironic-inspector (e.g. during the Victoria cycle ironic-python-agent is tested against stable/ussuri of ironic and ironic-inspector).

Third party CI jobs are expected to run on the intermediate branches the same way as they would on master. As soon as support for a specific branch is over, the 3rd party CI jobs may be turned off for it. Since we are only going to accept high and critical bug fixes to new branches, only minor load increase is expected on 3rd party CI systems.

Upgrades and Backwards Compatibility

See *Stable branches and upgrades* and *Testing*.

Documentation Impact

To make intermediate releases obviously consumable, we will need a new web site focused around standalone ironic. It will display the latest versions of the components and ironic-python-agent images, point at the way to consume them and provide documentation for each minor or major release.

The releasing documentation will be updated to follow this model.

References

5.15.169 No IPA to conductor communication

https://storyboard.openstack.org/#!/story/1526486

This spec intends to make agent->ironic communication optional, instead using polling to make all communication inbound to the agent.

Problem description

As part of the boot process IPA must call the ironic API to query its node ID, and notify ironic when its completed the boot process.

This implies that a target node has network access to the ironic API, which means that a malicious party could in theory attack the control plane from an instance. If the same control plane holds Keystone, Neutron, or other such services, the attacker can now DoS or compromise those. This grants them significant control over the infrastructure.

A deployer could mitigate this security flaw by using two networks for hosts:

- A provisioning network, which has access to the ironic API, but no ability to communicate with other nodes in the datacenter.
- A tenant network, which can communicate with the outside world, and other hosts, but cannot contact the ironic API.

However, this doesnt scale with medium to mega scale deployers who leverage layer 3 network topologies. In L3 networks a subnet is constrained to a single rack. This means that to leverage two networks to image hosts one would need to provision a second subnet for every single rack.

Compounding the issue further, different networks with different fundamental security policies implies that these disparate policies must be enforced. Thus, for each of your provisioning networks that require access to the ironic API, there must be access controls configured and enforced in the firewall.

In the context of hundreds, or thousands of racks, this does not scale.

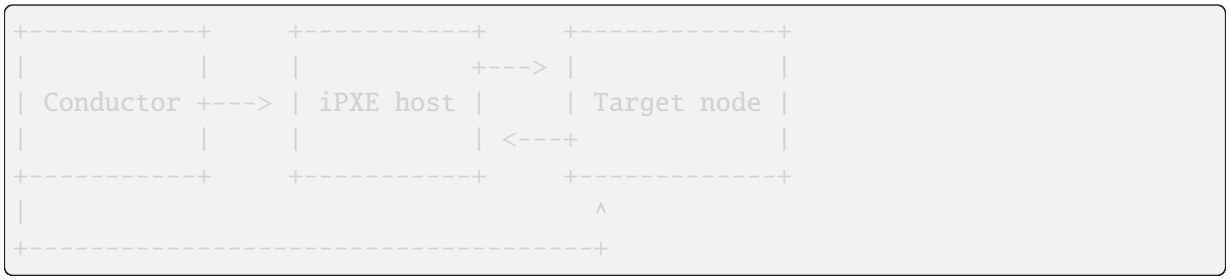
An example of the (potential) security problem:

- Bob boots a host with ironic, this host is publicly routable to the internet. Mallory finds and compromises this host. She then attacks the ironic API from this host. Once she compromises the ironic API, she then starts booting other hosts in the datacenter with a compromised disk image. If Bob uses ironic to manage every host in the datacenter, then Mallory has now effectively owned an entire datacenter.

To remediate this, we need to reduce the attack surface of the control plane by removing the need for the data plane to be able to send traffic to the API host. To do this, we need to be able to tell the agent that it should not call ironic, and make ironic poll the agent instead.

If the deployer runs DHCPD and serves PXE/iPXE from the control plane, then there is still logical network access between the target node and the control plane. However, this is easily fixed by running those services on dedicated intermediary hosts which do not have network access to the rest of the control plane.

In this context the network flow is:



And a simple example of the provisioning process would be:

1. Conductor receives call to boot node.
2. Conductor creates boot data, without the ipa-api-url parameter.
3. Conductor sends OOB call to power target node on.
4. Target node boots, uses DHCP to get IP.
5. Target node runs PXE/iPXE, pulls data from ipxe host.
6. Target boots and runs IPA.
7. Conductor polls for IPA until it is alive.

8. Conductor calls IPAs `get_hardware_info` command to get information about the nodes hardware. This is used to validate the MAC addresses to ensure this is the node we are expecting.
9. Conductor calls IPAs `node_info` command to give it the data it needs to do its job, including the config data returned by the lookup API.
10. Conductor calls IPA commands on target node to walk it through the provisioning process.
11. Conductor polls instance at a configurable interval to check on state, gather information, etc.
12. Target is complete. Conductor reboots target node.
13. Conductor cleans up boot data.
14. Complete.

An example of this as a fix to the security problem:

- Bob boots a host with ironic, this host is publicly routable to the internet. Mallory finds and compromises this host. She attempts to attack the ironic API. The connection times out. She gives up and attacks the iPXE host. She succeeds and compromises the iPXE host. She then attempts to attack the ironic API. The connection fails. Mallory, disappointed, gives up and puts her life of crime behind her.

In this case, even though Mallory has compromised the target node, there is no intrinsic network access between the target node and the control plane. Thus her only route of attack against the provisioning infrastructure would be DoS, or to impact hosts which are in the process of booting. But she has no ability to attack the queue, conductor, api, db, etc. She cannot gain control over the infrastructure, and her attack has been limited.

Proposed change

We will add two options to the [agent] category:

- `poll_only`: BoolOpt to enable passive mode. Defaults to False.
- `poll_interval`: IntOpt, poll interval in seconds. Defaults to the current `[api]/ramdisk_heartbeat_timeout` setting.

And one option to the [api] category:

- `disable_agent_api`: BoolOpt which disables the agent lookup and heartbeat APIs. Defaults to False.

If `poll_only` is enabled, we do not pass the `ipa-api-url` kernel command line parameter to IPA, which will disable the node lookup and heartbeat mechanisms.

If `poll_only` is enabled, the conductor will use a periodic task to query each agent at an interval as defined in poll_interval instead of querying the agent after a heartbeat is received. This periodic task will only query nodes in states IPA would normally be heartbeating in: (DEPLOY*, RESCUE*, CLEAN*).

It is assumed that the deployer should disallow communication between the target node and the ironic API. However, if an API call does come through when `disable_agent_api` is True, then Ironic should return a 403.

For this mode, we will also need to remove `ipa-api-url` being passed as kernel parameter to the agent.

We will also add a `node_info` command to IPA, described below, which the conductor will use to pass the lookup data to a node.

Last, we will add a `get_hardware_info` command to IPA, which will return hardware info we can use to ensure the node is the node we are expecting.

Notes:

- This spec depends on the assumption that ironic can look up the node IP in Neutron. Deployments without Neutron are not supported with poll_only=True. This may be added in the future.
- ironic-inspector is out of scope for this feature, as it does not use Neutron.
- There may be a use case to set `poll_only` per node, rather than globally. However, this is outside the scope of this spec.

Alternatives

None.

Data model impact

None

State Machine Impact

None

REST API impact

The lookup and heartbeat APIs used by agents will now return a 403 when `disable_agent_api` is set to True.

Client (CLI) impact

None

ironic CLI

None

openstack baremetal CLI

None

RPC API impact

None

Driver API impact

Deploy drivers will need to ensure that anything reaching into some agent can also be triggered by a periodic task.

Nova driver impact

None

Ramdisk impact

ironic-python-agent mostly supports this already, as it will run just fine without an API URL.

Some steps may also require other data returned by the lookup endpoint. Well add a new synchronous command `node_info`, which will take this data as a single `node_info` argument and store it in memory for later use. Ironic will call this command when it first notices that IPA is up.

To validate the node is the node we expect, well add another synchronous command `get_hardware_info`. This will return the MAC addresses at first, but could be evolved later to include things like serial numbers, etc.

Security impact

This change will prevent a malicious actor from using IPA as a vector of attack against the ironic API.

Note that TLS on the agent API is still important to completely secure the interactions between IPA and Ironic; however, this is outside the scope of this spec.

Other end user impact

None

Scalability impact

Polling target nodes for state from the conductor could have scale issues when managing many thousands of nodes. However, polling will be done in a thread pool, and so there should be limited impact.

Performance Impact

Polling in a large parallel fashion will introduce additional CPU load on the conductor nodes. Deployers may need to scale out their conductor nodes to handle the additional load.

Other deployer impact

Recap of the configuration options added:

[agent] * poll_only (type=BoolOpt, default=False) * poll_interval (type=IntOpt, default=<[api]/ramdisk_heartbeat_timeout>)

[api] * disable_agent_api (type=BoolOpt, default=False)

We should document where each of these needs to be set (API vs conductor hosts).

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
jroll

Other contributors:
penick

Work Items

- Enable IPA to skip the lookup process when ironic does not pass the `ipa-api-url` kernel parameter.
- Create the `get_hardware_info` IPA command.
- Create the `node_info` IPA command.
- Add the new options to ironic.
- Enable Ironic to use polling for agent actions/status rather than using the heartbeat as a trigger.
- Make ironic call the `node_info` command after IPA boots, when in polling mode.
- Disable heartbeating in the agent in polling mode.
- Test scale and performance impact on periodic tasks.
- Lots of documentation, especially in admin guides. It may also be worth a large blurb in the reference architecture guide.

Dependencies

None.

Testing

We should configure one of the existing tempest jobs to use this feature.

Upgrades and Backwards Compatibility

The deployer must update IPA in their images to support passive mode prior to upgrading Ironic and enabling the feature. If they do not, all imaging attempts will fail.

Documentation Impact

This feature needs to be documented as a deployment option.

The ironic-inspector docs need to be updated to capture that inspector wont work with poll_only=True.

Admin docs should be updated to note that firewall rules need to be implemented to actually close network access between the target node and the ironic API.

References

None

5.15.170 Support node history

https://storyboard.openstack.org/#!/story/2002980

This spec proposes node history support for nodes, which is useful for identifying issues.

Problem description

Currently ironic uses one last_error field to record error information when an operation failed, this field is easily overwritten, to traceback the root cause we have to search logs on the conductor host located somewhere in the cloud. To make bare metal management easier, it would be handy to have a history, especially, errors and state transitions of a node.

The proposal is to introduce a new table to store those events and provide API support to retrieve them.

Proposed change

Introduces a new table named `node_history` and a db object `NodeHistory`, see *Data model impact* for the schema definition.

Implements API layer to support node history query. The node history is supposed to be query only.

Only two kinds of events will be logged in this proposal:

- State transitions
- Everything goes to last_error, this also covers node maintenance state change.

The range could be extended according to requirements in the future, but not included in this spec.

Introduces a periodic task to remove node history entries which exceed specified maximum of number, the number will be configurable by configuration options.

Adds a `history` module to provide history interface abstraction and provides two implementation with `none` and `database`.

Alternatives

Other solutions exist, like using LOG collector and aggregator, but they need more integrations and not directly supported from ironic.

Data model impact

A new database table will be added with following schema:

```
op.create_table('node_history',
                sa.Column('created_at', sa.DateTime(), nullable=True),
                sa.Column('updated_at', sa.DateTime(), nullable=True),
                sa.Column('id', sa.Integer(), nullable=False),
                sa.Column('uuid', sa.String(length=36), nullable=False),
                sa.Column('conductor', sa.String(length=255), nullable=True),
                sa.Column('event', sa.Text(), nullable=True),
                sa.Column('node_id', sa.Integer(), nullable=True),
```

(continues on next page)

(continued from previous page)

```
                sa.Column('user'), sa.String(length=32), nullable=True),
                sa.PrimaryKeyConstraint('id'),
                sa.UniqueConstraint('uuid', name='uniq_history0uuid'),
                sa.ForeignKeyConstraint(['node_id'], ['nodes.id'], ),
                                        mysql_ENGINE='InnoDB',
                                        mysql_DEFAULT_CHARSET='UTF8')
                sa.Index('node_id', 'node_id')
```

`event` is the string conveys what happened to the node, the content will be truncated to 1000 characters.

`conductor` is the hostname of the conductor who recorded the entry.

`user` is the requester for the operation from the context, for the Identify service its a string with fixed length.

State Machine Impact

None

REST API impact

Following endpoints will be added to support querying node history, microversioned. Clients with earlier microversion will receive 404.

- GET /v1/{node_ident}/history
  - Retrieve the list of events logged for this node. By default `uuid`, `event` and `created_at` are returned. The `event` will be truncated to 255 to give a brief information. Detailed history entry will be returned if `detail` is set to True in the query string.
  - For a normal request, 200 is returned.
- GET /v1/{node_ident}/history/{history_uuid}
  - Get detailed information of an event.
  - For a normal request, 200 is returned.

Client (CLI) impact

openstack baremetal CLI

OSC will be enhanced to support following operations:

- `openstack baremetal node history list`: list all events kept for this node
- `openstack baremetal node history show` <uuid>: show a specific node event

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

None

Security impact

None

Other end user impact

None

Scalability impact

Node events could occupy considerable amount of data in the database when this feature is enabled, depending on the scale of bare metals and activities. In such case the configuration options of this feature should be evaluated.

Performance Impact

The new periodic task and database access will use some resource, but should be trivial.

Other deployer impact

Adds following configuration options to change the behavior of this feature:

- `[conductor]node_history_backend`: can be `none` and `database`. `none` does nothing and effectively disable this feature, this is the default.
- `[conductor]node_history_max_entries`: how many events ironic should keep. Oldest events will be removed when reached max entries. The default is 300, the minimum value is 1.
- `[conductor]node_history_cleanup_interval`: the interval in seconds, the clean up periodic task should be scheduled. One day by default. Set to 0 will disable periodic clean up.
- `[conductor]node_history_cleanup_batch_num`: the maximum number of entries will be removed during one clean up operation.

Developer impact

Other events could be added once this spec is implemented.

Implementation

Assignee(s)

Primary assignee:
<kaifeng, kaifeng.w@gmail.com>

Other contributors:
<None>

Work Items

Implements proposed work:

- Database support
- The history module and two backends namely none and database
- Log history at proper code path
- API support
- CLI support
- Documentation

Dependencies

None

Testing

The feature will be covered by unit test.

Upgrades and Backwards Compatibility

This feature is disabled by default.

Documentation Impact

Documentation will be updated.

References

None

5.15.171 Allow Leasable Nodes

https://storyboard.openstack.org/#!/story/2006506

A large bare metal deployment may consist of hardware owned by multiple owners who lease their nodes to users - lessees - who gain temporary and limited access to specific nodes. Ironic understands the concept of a hardware owner: a node can set its `owner` field to a project, and that project can gain API access to that node through the use of an updated policy file. However, Ironic does not understand the concept of a node lessee.

This spec describes a solution that accommodates the notion of a node lessee.

Problem description

Ironic currently supports two classes of users: administrators, who control the entire inventory of hardware; and owners, who have policy-limited API access to the nodes that they own. However, Ironic does not support non-administrative users - users who can deploy upon a node, and perhaps have access to an extremely limited subset of the API (such as node power functions).

Proposed change

Node lessees can be supported with the following changes:

- Add a new `lessee` field to the node object. This field must either be empty or set to a Keystone project id.
- Update the node controller so that policy checks pass in a nodes `lessee` information. Note that Ironic already does that with node owners[0].
- Update Ironics default generated policy file to include an `is_node_lessee` rule:
  - is_node_lessee: project_id:%(node.lessee)s

  The remainder of the policy file will stay the same, so that there is no change to default API access.
- Update the node list function so that projects with access to `baremetal:node:list` are returned nodes with matching `owner` or `lessee` fields.
- Update Ironic allocations so that allocations with owners can match nodes by a nodes `owner` or `lessee`.

Note that this work does not add any new scheduling responsibilities in Ironic. A new Nova filter, such as an updated version of the proposed NodeOwnerFilter[1], would be desirable; and Blazar could integrate with the `lessee` field as they see fit. However, the proposed work does integrate well with the existing ability to create a restricted allocation.

Further down the line when Ironic creates a Deployment API, we can have the new Deployment API actions default to being accessible to node lessees.

Alternatives

Lessee information could be stored in a dictionary field such as `properties` or `extras`. However this makes updating database queries far more difficult, and the non-administrative user concept feels distinct enough to warrant a new field.

Data model impact

A `lessee` field will be added to the nodes table as a `VARCHAR(255)` with a default value of `null`.

State Machine Impact

None

[0] https://opendev.org/openstack/ironic/src/branch/master/ironic/api/controllers/v1/utils.py#L1178
[1] https://review.opendev.org/#/c/697331/

REST API impact

- A `lessee` field will be returned with the node object.
- The REST API will pass in the `lessee` for node policy checks.
- The API will be updated to allow a user to set/unset the value through the API.
- The node list API will be updated to allow filtering by `lessee`.
- The limited `baremetal:node:list` action will be updated to match nodes by both lessee and owner.
- A new API microversion will be introduced for the new node `lessee` field.

Client (CLI) impact

None

ironic CLI

None

openstack baremetal CLI

An update will be needed to enable a user to set/unset `lessee` from the command line.

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

None

Security impact

This change allows functionality to be exposed to additional users. However this access is blocked off by default; it requires an update to the Oslo policy file, and can be adjusted as an administrator desires.

Other end user impact

None

Scalability impact

None

Performance Impact

Some functionality that previously matched nodes by `owner` will now have to match both `owner` and `lessee`. This should be doable at the database query level.

Other deployer impact

None

Developer impact

None

Implementation

Assignee(s)

Primary assignees: * tzumainn - tzumainn@redhat.com

Work Items

- Add database field.
- Add object field.
- Add REST API functionality and microversion.
- Update REST API documentation.
- Update python-ironicclient.
- Update node controller.
- Update allocations conductor.
- Write tests.
- Write documentation detailing usage.

Dependencies

None

Testing

We will add unit tests and Tempest tests.

Upgrades and Backwards Compatibility

The `lessee` node field will be created as part of the upgrade process with a default value in the database schema. This change has no end-user impact if the policy file is not updated.

Documentation Impact

REST API documentation will be updated.

References

5.15.172 Allow Node Owners to Administer Nodes

https://storyboard.openstack.org/#!/story/2006506

This spec describes an update that allows a node owner to administer their node through the Ironic API without being able to administer the entire node cluster. This is accomplished by exposing owner node data in the REST API for policy checks.

Problem description

Ironic is not multi-tenant; anyone with API access to one node has access to all nodes. While nodes have an `owner` field, it is purely informational and not tied into any form of access control.

Bringing full multi-tenant support to Ironic would allow us to address the following user stories:

Standalone Ironic

As users of a shared data center, we would like to use Ironic to manage our hardware resources. Each work group should be able to control their own resources without having access to hardware resources owned by other groups.

Data center operator

As the operator of shared datacenter, I would like to delegate power control of baremetal hardware to the hardware owners using the Ironic API. By using the Ironic API instead of existing embedded management protocols (such as IPMI) I can maintain the simplicity of a shared management network while having granular control over who can access what hardware.

Proposed change

The Ironic API already has a way of controlling access to the REST API: a policy engine[0] that is used throughout the REST API. However, when the node controller uses the policy engine[1], it does so without passing in any information about the node being checked. Other services such as Nova[2] and Barbican[3] pass in additional information, and we propose doing the same. In summary, we would like to:

- Assume that a nodes `owner` field is set to the id of a Keystone project. We refer to this project as the node owner.
- Update the node controller so that policy checks also pass in information about the node, including the `owner` field.
- Update Ironics default generated policy file to include an `is_node_owner` rule:
  - is_node_owner: project_id:%(node.owner)s

[0] https://github.com/openstack/ironic/blob/master/ironic/common/policy.py

[1] https://github.com/openstack/ironic/blob/master/ironic/api/controllers/v1/node.py#L225 Example of a current policy check. Note the use of `cdict`; it is being passed in as both the `target` and the `creds`.

[2] https://github.com/openstack/nova/blob/master/nova/api/openstack/compute/servers.py#L648-L652 Example of Nova creating a `target` dictionary.

[3] https://github.com/openstack/barbican/blob/stable/rocky/barbican/api/controllers/__init__.py#L59-L72 Example of Barbican creating a `target` dictionary.

The remainder of the policy file would stay the same, meaning that there is no change to default API access.

- Create documentation explaining how to update an Ironic policy file to give API access to owners. For example:
  - baremetal:node:set_power_state: rule:is_admin or rule:is_node_owner

  Note that Nova grants API access in a similar manner[4].

This update is enough to control access for most API functions - except for list functions. For those, we propose the following:

- Right now, the policy rule baremetal:node:get is used for both `get_all` and `get_one`. We can add two new policy rules: `baremetal:node:list_all` for retrieving all nodes, and `baremetal:node:list` for retrieving nodes whose `owner` field matches the querying project.
- Updating the REST API node list functions to first perform a policy check against `baremetal:node:list_all`. If it passes, then we return all nodes; otherwise we perform a policy check against `baremetal:node:list`. If that passes, then we return nodes filtered by the `owner` field against the project specified in the request context. These list functions already have the option of filtering by `owner`[5].
- Updating the default generated policy file to set `baremetal:node:list_all` and `baremetal:node:list` to `rule:baremetal:node:get` to ensure backwards compatibility.

Allocation API changes

Allocation objects represent a request to find and reserve a suitable node, and as such should be available for non-administrator users. For this purpose we will introduce an `owner` field for allocations and start distinguishing two types of allocations:

unrestricted allocations
: work the same as before. They can be created only by administrative users and can have any `owner` set. Creating them is governed by the existing policy `baremetal:allocation:create`.

restricted allocations
: are created by non-administrative users and have `owner` matching the project ID of a creating user. Creating them will be governed by a new policy `baremetal:allocation:create_restricted`.

The two policies will be exclusive and work as follows:

1. First, `baremetal:allocation:create` is checked. If it passes, then any value of `owner` is accepted. If no `owner` is provided, the field is left as `None`.
2. If the first check does not pass, `baremetal:allocation:create_restricted` is checked. The `owner` field must match the project ID of the user or be empty (in which case it is set to the project ID of the user). HTTP Forbidden is returned if:
   1. The project ID of the user cannot be determined (i.e. restricted allocations do not work in the standalone mode).
   2. The requested non-empty owner does not match the project ID.

[4] https://github.com/openstack/nova/blob/master/nova/policies/base.py#L27-L30 Example of Nova defaulting a rule that uses information from a `target` dictionary.

[5] https://github.com/openstack/ironic/blob/master/ironic/api/controllers/v1/node.py#L1872

3. Otherwise, creating the allocation fails.

Finally, when processing an allocation with non-empty `owner`, only nodes with a matching `owner` are considered.

Alternatives

One alternative is to perform these checks at the database API level. However that requires a new code pattern to be used on a large number of individual functions.

Data model impact

None

State Machine Impact

None

REST API impact

See details in Proposed change above.

Client (CLI) impact

None

ironic CLI

None

openstack baremetal CLI

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

None

Security impact

This change allows functionality to be exposed to additional users. However this access is blocked off by default; it requires an update to the Oslo policy file, and can be adjusted as an administrator desires.

Other end user impact

None

Scalability impact

None

Performance Impact

None: although node data needs to be retrieved in order to pass that information into a policy check, the controller functions already fetch that information.[6]

Other deployer impact

None

Developer impact

None

Implementation

Assignee(s)

Primary assignees: * tzumainn - tzumainn@redhat.com * larsks - lars@redhat.com

Other contributors: * dtantsur - dtantsur@redhat.com

Work Items

- Update node controller.
- Add documentation.
- Write tests.

Dependencies

None

Testing

We will add unit tests and Tempest tests.

[6] https://github.com/openstack/ironic/blob/master/ironic/api/controllers/v1/node.py#L227

Upgrades and Backwards Compatibility

Existing Ironic installations that use the `owner` field for something other than a project ID will be minimally affected for two reasons:

- If the `owner` field does not match a project ID (or is None), the proposed update to the policy file will not give any non-admin access to the Ironic API.
- This change has no end-user impact if the policy file is not updated. An existing install can simply choose not to update their policy file.

Documentation Impact

We will include additional documentation describing the possible applications of using the `node_owner` policy roles.

References

5.15.173 Add node resource_class field

https://bugs.launchpad.net/ironic/+bug/1604916

Nova has a plan for scheduling ironic resources the same way as other nova resources, that involves making each ironic node a resource provider, which has a resource class. That resource class is referenced by the nova flavor, in short saying that the flavor requires exactly one thing from that resource class. When running the resource tracker, nova must be able to associate each ironic node with a resource class, which operators must be able to specify (because operators are the people creating the flavors and need to know how to associate the flavor). To allow for this, we add a new resource_class field to ironics node object.

Problem description

Currently, nova tracks ironic resources with a (host, node) tuple. This causes a number of problems within novas internals, and the nova team is trying to get away from this. At the same time, nova wants to schedule ironic instances the same way as other instances are scheduled. This allows ironic to follow along in the ongoing scheduler changes, which will help reduce (or eventually eliminate) scheduling races, and benefit from other scheduler optimizations coming down the road (like making qualitative decisions more performant).

Proposed change

The current proposal in nova is to make each ironic node a resource provider, which is associated with a resource class.[0] A nova flavor may declare that it requires some amount of a given resource class; in ironics case it would require one and only one. The resource provider record for an ironic node will declare that it provides exactly 1 (or 0, if it is in maintenance or similar) of the resource class for that provider record.

This proposal still allows nodes to be scheduled to based on qualitative properties, such as capabilities or affinity rules. The resource class is simply the first filter in this case (though it isnt a traditional scheduler filter).

To do so, nova needs to know which resource class the resource provider record needs to be associated with. As operators manage the flavors that will be linked to these classes, we need a way for the operator to specify what class each node is in, so that the flavor is linked correctly. As such, we add a resource_class field to the node record in ironic, which nova will use when creating the resource provider record.

As an example, imagine an ironic deployment has the following nodes:

```
- node-1:
  resource_class: ironic-gold
  properties:
    cpus: 4
    memory_mb: 16384
    capabilities:
      boot_mode: uefi,bios
- node-2:
  resource_class: ironic-silver
  properties:
    cpus: 2
    memory_mb: 8192
```

The operator might define the flavors as such:

```
- baremetal-gold
  resources:
    ironic-gold: 1
  extra_specs:
    capabilities: boot_mode:bios
- baremetal-gold-uefi
  resources:
    ironic-gold: 1
  extra_specs:
    capabilities: boot_mode:uefi
- baremetal-silver
  resources:
    ironic-silver: 1
```

Note that the flavor definition is a strawman and may not be the actual keys used when this is implemented.

A nova user booting an instance with either the baremetal-gold or baremetal-gold-uefi flavor would land on node-1, because capabilities can still be passed down to ironic, and the resource_class on the node matches what is required by flavor. The baremetal-silver flavor would match node-2.

Alternatives

Keep doing the (host, node) thing long enough such that the nova team decides they want to remove our driver from nova, and subsequently remove the (host, node) tuple and break our driver horribly.

Data model impact

Adds a resource_class field to the nodes table. This will be a VARCHAR(80) because that matches novas table structure. There will be data migrations provided for this change. The objects API will also take this change, with a corresponding version bump. This will default to NULL.

State Machine Impact

None.

REST API impact

The new resource_class field will be exposed in the API, just like any other string field, with the same semantics.

Default policy for this field should be the same as driver, network_interface, etc.

Filtering and sorting on this field will be added.

There will be a microversion bump, as usual.

Client (CLI) impact

The client will be updated to add the field.

ironic CLI

The CLI will be updated to add the field.

openstack baremetal CLI

The CLI will be updated to add the field.

RPC API impact

Only the objects changes mentioned above.

Driver API impact

None.

Nova driver impact

Immediately, well pass the *resource_class* field back up to the resource tracker, so that nova can put these resources in the placement database in Newton. There will be a small patch that bumps the API version were using and passes the field back in the resource_dict. This will need a release note dictating the ironicclient version and available API version needed to run this code.

During Ocata, work will be done on the scheduler to use this for scheduling, however that is outside of the ironic driver.

Ramdisk impact

None.

Security impact

None.

Other end user impact

None.

Scalability impact

None.

Performance Impact

None.

Other deployer impact

After the deployment of the Newton version of nova, deployers will need to populate the resource_class field in ironic, and associate the flavors, before deploying the Ocata version of nova.

Developer impact

None.

Implementation

Assignee(s)

jroll

Work Items

- Add the field to the DB.
- Add the field to the objects model.
- Add the field to the API, with filtering and sorting.
- Doc updates for install guide and such.

Dependencies

None.

Testing

Unit tests should suffice here.

Upgrades and Backwards Compatibility

No direct impact, but its important to note that scheduling in the Ocata version of nova will still fall back to the old method, if resource_class is set to NULL for any node. Operators should have up until the P version of nova to populate this data.

Documentation Impact

Add the new field to the API reference.

Well also need to update the install guide and any other docs that talk about setting up nova with ironic, to make sure that deployers are setting this field when adding nodes. This will also need to be communicated extremely hard via release notes (and probably ops list emails).

References

[0] https://review.opendev.org/#/c/312696

5.15.174 Support for node retirement

https://storyboard.openstack.org/#!/story/2005425

This spec proposes to add initial support for retiring nodes from ironic.

Retiring nodes is a natural part of a servers life cycle, for instance when the end of the warranty is reached and the physical space is needed for new deliveries to install replacement capacity.

However, depending on the type of the deployment, removing nodes from service can be a full workflow by itself as it may include steps like moving applications to other hosts, cleaning sensitive data from disks or the BMC, or tracking the dismantling of servers from their racks.

In order to help Deployers with these tasks, ironic should provide basic means to mark nodes as being retired, i.e. mark them as not eligible for scheduling any longer (while still allowing other operations such as cleaning), or support a convenient search for and list such nodes.

Extensions of this initial support, such as extending the state machine by an explicit retired state (which could be used as a starting point for a dedicated retirement workflow with end-of-life cleaning or other more elaborate actions, such as prepare for donation) are beyond the scope of this spec.

Problem description

There is currently no explicit support in ironic for node retirement: when a node needs to be taken out of service, the instance is deleted (which triggers cleaning) and the node is deleted, either directly from state available or after being moved to manageable (and potential additional cleaning).

There are at least two issues that this spec tries to fix:

- between the deletion of the instance and the deletion of the node, there is a time window in which a new instance could be scheduled to a node on its way into retirement (setting maintenance to True does only partially help with this scheduling race as apart from not being meant to be used in this situation this will prevent cleaning);
- there is no easy way to mark nodes as retired; this makes identifying and listing such nodes (as required by third party tools or the remaining retirement workflow) cumbersome.

The retirement use case is different from other use cases, like a non-functional node which needs to be taken out of service for some time, in that nodes may be marked for retirement a long time before anything actually happens on the node. Also, retired nodes are not supposed to enter service again.

Proposed change

The proposal is to extend the list of node properties by two new fields:

- retired (True/False)
- retired_reason (text)

in analogy to the existing protected and protected_reason.

The retired field shall signal that a node is meant to be retired and that the node should not be considered for scheduling any longer. The retired field can be set irrespective of the nodes state, in particular when the node is active.

Active nodes which are cleaned while retired is True, e.g. upon instance deletion, go to manageable (rather than available). This leaves no window where a retired node would receive another instance. Otherwise, retired set to True shall not interfere with cleaning or rebuilding.

Nodes with retired set to True cannot move from manageable to available (to prevent accidental reuse): the provide verb is blocked. In order to move these nodes to available, the retired field needs to be set to False first.

The new field shall also be used to get these nodes quickly via a list command, e.g. by an additional flag retired (in analogy to maintenance).

Alternatives

An alternative to address the lack of support for node retirement would be to introduce a new state retired in the ironic state diagram. While this would require additional efforts to implement, there are no obvious benefits compared to the proposal of this specification.

Using the currently available means in ironic, such as the maintenance state and maintenance_reason, is certainly possible but will cause inconveniences during node cleaning and when trying to extract the list of nodes to be removed.

Data model impact

The nodes table will get two additional fields:

- retired (tinyint)
- retired_reason (text)

For existing instances, i.e. for data migration, retired will be set to False and retired_reason will be left empty.

State Machine Impact

The retired field can be set on nodes in any state except available. Nodes in available should be moved to manageable first to ensure backwards compatibility with tools like metalsmith. This also eliminates the need for changes in the nova driver.

The retired field can also be set for nodes in transient states, as long as the node is not locked.

When retired is set to True, a node will not move to available after cleaning, but to manageable.

When retired is set to True, a node cannot move from manageable to available, the corresponding provide verb will return an HTTP 409.

REST API impact

The REST API will need to be extended with a new version to include the new retired field, in a similar way as the protected field was introduced [0][1].

In addition, there should be a filter /v1/nodes?retired=True/False to easily identify retired nodes.

Client (CLI) impact

ironic CLI

None.

openstack baremetal CLI

The retired and retired-reason options will be added to the set and unset subcommands:

- openstack baremetal node set retired retired-reason <reason> <node>
- openstack baremetal node unset retired <node>

shall be added to set and unset a nodes retired field (and provide a reason in the case of set).

An additional flag retired shall be added to the openstack baremetal node list command to restrict the returned result to the nodes which have the retired flag set.

RPC API impact

Driver API impact

None.

Nova driver impact

None.

Ramdisk impact

None.

Security impact

None.

Other end user impact

None.

Scalability impact

None.

Performance Impact

None.

Other deployer impact

None.

Developer impact

None.

Implementation

Assignee(s)

Primary assignees:
Arne Wiebalck (arne_wiebalck) Riccardo Pittau (rpittau)

Other contributors: None

Work Items

- Add new database fields
- **Adapt state machine if retired is set to True**
  - change cleaning > manageable
  - move available > manageable
  - block manageable > available
- Exclude retired nodes from periodic tasks
- Extend API
- Extend cli
- Add new API to openstacksdk
- Add documentation

Dependencies

None.

Testing

Needs testing similar to maintenance.

Upgrades and Backwards Compatibility

Upon upgrades the new fields need to be set as specified in the Data model impact section.

Documentation Impact

The additional retired field and its intended use need to be documented.

References

[0] https://storyboard.openstack.org/#!/story/2003869 [1] https://review.opendev.org/#/c/611662

5.15.175 Node Traits

https://bugs.launchpad.net/ironic/+bug/1722194

Add more granular Nova placement of ironic nodes by allowing operators to set a list of traits associated with each node.

Problem description

While the recent addition of Resource Class on every node has helped improve the way ironic resources are scheduled in Nova, using the new Placement API, there are many use cases that need more granular control.

An example use case is dedicating a pool of ironic nodes for a specific project. Another is allowing flavors that best fit to the available hardware, you might want smaller flavors to build on larger machines if you are out of smaller machines.

In a similar way, you might have a pool of hardware that are all the same except for the chosen RAID configuration. Longer term, when you provision the node via Nova boot it would be good to be able to automatically reconfigure the hardware with the chosen RAID configuration. For now this is considered out of scope.

Proposed change

The Placement API has the ability to tag a resource provider with a trait: http://specs.openstack.org/openstack/nova-specs/specs/pike/approved/resource-provider-traits.html

The nova ironic driver currently ensures a Resource Provider exists for every Ironic Node. It already populates the inventory with the correct quantitative resources (i.e the Resource Class).

To get the extra granularity of scheduling we depend on adding traits to the resource provider, and the implementation of the following Nova specs that ensure the traits requested in flavors are honored by the scheduling process:

- http://specs.openstack.org/openstack/nova-specs/specs/queens/approved/request-traits-in-nova.html
- http://specs.openstack.org/openstack/nova-specs/specs/queens/approved/add-trait-support-in-allocation-candidates.html

The current proposal is for Ironic to store a list of traits for each node. That list can then be synced into the appropriate resource provider in the Placement API by Novas ironic virt driver, in a similar way to how the RESOURCE_CLASS of each ironic node is reported today.

Lets talk about the use case of dedicating specific Ironic nodes for use only by a specific set of projects. The remainder of the hosts are for general use. If a user has a dedicated pool of resources, they have the ability to pick if they create an instance in their dedicated pool or in the general pool. Other users are only able to build in the general pool. One way to bisect the nodes like this is assigning traits such as *CUSTOM_IRONIC_NODE_PROJECT_B* and *CUSTOM_IRONIC_NODE_GENERAL_USE* to the appropriate ironic nodes. Then there is a public flavor to target the general pool of hosts, and a private project specific flavor that targets their dedicated pool. By taking this approach it is easy to add additional pools of nodes for other sets of projects. It is easier because you dont need to modify any of the existing nodes and flavors when you add an additional pool of nodes.

Note that inspection rules can be used to set the initial value for a nodes Resource Class. It is expected the initial set of traits for a node can be set in the same way.

Note

There is already a Nova spec proposed discussing the Nova side of how we can assign traits to the resource providers, and on Nova boot send the chosen traits back to ironic:

- http://specs.openstack.org/openstack/nova-specs/specs/queens/approved/ironic-driver-traits.html

When provisioning a node, the Ironic virt driver in Nova will now send additional flavor extra_specs to Ironic. Currently only extra_specs starting with *capabilities* are set in instance_info. After the above spec, the ironic virt driver will also include the flavor extra specs that define what traits have been requested, storing them *instance_info[traits]*.

Note

Note there is no change to how capabilities are used as part of this spec.

Ironic needs to validate that *instance_info[traits]* is consistent with the list of traits currently associated with Ironic node, i.e. we must check that *instance_info[traits]* only includes traits that are already in the list of traits set on the Ironic node. This is particularly useful in preventing strange behaviour due to races between the update of an Ironic nodes trait list and getting that list copied into the Placement API by the Ironic virt driver. This validation is probably best integrated as part of `driver.deploy.validate` or similar, to ensure it is triggered by a call to deploy a node and a call to validate a node.

Note

The ironic virt driver in Nova will sync the list of traits reported by each Ironic Node into Novas Placement API, in a similar way to how the Resource Class is done today. Should an operator talk directly to Placement to adjust the traits, this process will remove all those modifications when the next sync occurs. Operators must use the Ironic API to change the list of traits on a given node. Ironic is assumed to be the source of truth for the list of traits associated with a Resource Provider that is representing an Ironic node.

Alternatives

In the future, it is expected that a driver may need to validate some well known traits to see if they are supported. In addition it may be that some drivers automatically report some traits. Following on from that, it could be that drivers read the traits specified in instance_info to reconfigure the node. This is all out of scope here.

Looking at what is in scope, there are alternative approaches:

- Make operators (and ironic-inspector) talk directly to the Placement API to set the node traits. This would be very odd given the Nova sync of the Resource Providers using the ComputeNode uuid as the uuid for the Resource Provider. You cant set the traits until that sync has completed. Right now placement is largely an internal API with few policy controls, so that would all need to change to allow the above. It all seems very messy.
- Keep traits APIs in Ironic, but make it a pass-through proxy to Placement. This is would make Ironic hard depend on placement, which would be a strange requirement for things like Bifrost, and would complicate upgrades.

The current approach keeps Ironic from needing to depend on placement in any way, which works well.

The REST API described below allows fine grained control of setting traits, following the patterns from the API-WG and existing placement APIs. We could instead have just extended the existing ironic node PATCH interface.

The suggested CLI approach below follows the new API, but as an alternative we could have extended the existing CLI interface around the PATCH API call. It would have looked something more like:

- openstack baremetal node set trait CUSTOM_FOO <node-uuid>
- openstack baremetal node unset trait CUSTOM_FOO <node-uuid>

Data model impact

Following a similar pattern to the existing table for tags, we will need to add this new table to store the traits associated with a node:

```
CREATE TABLE node_traits (
    node_id INT(11) NOT NULL,
    trait VARCHAR(255) CHARACTER SET utf8 NOT NULL,
    PRIMARY KEY (node_id, trait),
    KEY (trait),
    FOREIGN KEY (node_id)
      REFERENCES nodes(id)
      ON DELETE CASCADE,
)
```

A new `ironic.objects.traits.NodeTraitList` object will be added to the object model. The ironic.objects.traits.NodeTraitList field in the python object model will be populated on-demand (i.e. not eager-loaded).

A trait should be defined in a way that matches the placement API definition, as a Unicode string no longer than 255 characters.

State Machine Impact

No impact.

REST API impact

The placement API defines a set of standard traits in the *os-traits* library. Any traits that are not defined in that library must start with the prefix of *CUSTOM_*. Any trait set in Ironic must follow these rules, else the ironic Nova virt driver will be unable to add the traits in Placement. For similar reasons there is a limit of 50 traits on any node, to match the limit in Placement. A request to add a badly formatted trait should get a response with the status code 400.

Note at no point does Ironic talk to the Placement API. The above validation depends only on access to the python library *os-traits*. As such, this validation poses little restriction on how traits can be used in standalone Ironic to assign arbitrary traits on particular Ironic nodes. Any non-standard traits simply need to have a prefix of `CUSTOM_` added. For more details on *os-traits* please see: https://docs.openstack.org/os-traits/latest

For convenience, it will be possible to get the full list of nodes and the traits associated with each node by extending the existing API in the following way (when requesting a high enough microversion that

includes these details):

```
GET /v1/nodes/detail

{
  "nodes": [
    {
      ...
      "traits": ['CUSTOM_FOO', 'CUSTOM_BAR', 'CUSTOM_BAZ'],
      ...
    }
  ]
}
```

In a similar way to other fields, we will also support a request to get just this field (in part to make the Nova virt driver polling more efficient):

```
GET /v1/nodes/?fields=uuid,traits

{
  "nodes": [
    {
      "uuid": "uuid-1",
      "traits": ['CUSTOM_FOO', 'CUSTOM_BAR', 'CUSTOM_BAZ']
    },
    ...
  ]
}
```

The manipulation of node traits will follow the patterns established by both the placement API and API WG tags spec:

- https://developer.openstack.org/api-ref/placement/#resource-provider-traits
- http://specs.openstack.org/openstack/api-wg/guidelines/tags.html

To start with there will be a new traits resource that follows the above patterns.

Example request for all node traits:

```
GET /nodes/{node_ident}/traits
```

Response:

```
{
    "traits": ['CUSTOM_FOO', 'CUSTOM_BAR', 'CUSTOM_BAZ']
}
```

Example request to set all node traits to given list:

```
PUT /nodes/{node_ident}/traits
{
    "traits": ['CUSTOM_FOO', 'CUSTOM_BAR', 'CUSTOM_BAZ']
}
```

Response:

```
{
    "traits": ['CUSTOM_FOO', 'CUSTOM_BAR', 'CUSTOM_BAZ']
}
```

The response on success is status code 200. On failure to validate (using the os-traits library) we return the status code 400 (Bad Request), matching the HTTP Guidelines from the API-WG.

Note that unlike with Resource Class, we are allowing the trait to be updated at any time. This is mostly because placement allows such updates and because although the Resource Class and Ironic node are used in the allocations in placement, traits are not used in allocations.

In a similar way the following API removes all the traits:

```
DELETE /nodes/{node_ident}/traits
```

The response on success is status code 204, with an empty body.

To add or remove an individual trait use:

```
PUT /nodes/{node_ident}/traits/CUSTOM_FOO
<no body>

DELETE /nodes/{node_ident}/traits/CUSTOM_FOO
```

Filtering the node list by traits should work as expected:

```
GET /nodes?traits=CUSTOM_RED,CUSTOM_BLUE
GET /nodes?not-traits=CUSTOM_RED,CUSTOM_BLUE&traits=CUSTOM_FOO
GET /nodes?traits-any=CUSTOM_RED,CUSTOM_BLUE
GET /nodes?not-traits-any=CUSTOM_RED,CUSTOM_BLUE
```

As mentioned above, the final change that is made is to ensure `instance_info/traits` is a subset of the traits set on the Ironic node. This should be part of the existing `driver.deploy.validate()` call (or similar) such that the traits will be checked both before a deploy starts and on an explicit node validate call.

Client (CLI) impact

ironic CLI

No changes, it is deprecated.

openstack baremetal CLI

You can list the traits on a node:

- openstack baremetal node list fields uuid name traits
- openstack baremetal node show <node-ident> fields uuid name traits
- openstack baremetal node trait list <node-ident>

You can update the list of traits on a node:

- openstack baremetal node add trait <node-ident> CUSTOM_FOO CUSTOM_BAR

- openstack baremetal node remove trait <node-ident> CUSTOM_FOO CUSTOM_BAR
- openstack baremetal node remove trait all <node-ident>

This is roughly copying the command syntax of consistency groups: https://docs.openstack.org/python-openstackclient/latest/cli/command-objects/consistency-group.html#consistency-group-add-volume

It is common to use set and unset for key value pairs, but add and remove seems a better fit for in-place modifications of a list. It stops any ambiguity of set meaning either an addition of a list of traits or replacing the whole list of traits. Another alternative is to add trait operations into the existing `openstack baremetal node set` operation, but we are instead following the structure of the API.

You can query the list of nodes using traits:

- openstack baremetal node list trait CUSTOM_RED not-trait CUSTOM_BLUE
- openstack baremetal node list trait-any CUSTOM_RED CUSTOM_BLUE
- openstack baremetal node list not-trait-any CUSTOM_RED CUSTOM_BLUE

RPC API impact

No impact.

Driver API impact

No impact.

Nova driver impact

Need to ensure the correct flavor extra specs are passed back when starting a node.

Ramdisk impact

None

Security impact

There will be a hard coded limit of 50 traits for any Node to prevent misuse of the API. This prevents denial of service attack where the database is filled up by a rogue user setting lots of traits. Really the limit is in place to match the limit applied in the placement API.

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

None

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
John Garbutt (johnthetubaguy)

Other contributors:
Dmitry Tantsur (dtantsur) Mark Goddard

Work Items

- Add table to store traits for nodes
- Add object to expose the table
- Add new traits API
- Add openstack cli support for the new API
- Follow up with Nova driver work

Dependencies

The following nova spec depends on this spec:

- http://specs.openstack.org/openstack/nova-specs/specs/queens/approved/ironic-driver-traits.html

Testing

Nova functional tests are planning on covering the scheduling aspects of the integration. As part of this spec we will focus on ensuring the API works correctly to persist the traits for given nodes, and query resources using traits.

Upgrades and Backwards Compatibility

Longer term, capabilities and other APIs may be phased out. We are not considering that as part of this spec. There is much more work needed before we have feature parity between the old and new scheduling mechanisms.

Documentation Impact

Need to update the API-REF and the admin doc to cover how to use the new API.

References

- http://specs.openstack.org/openstack/nova-specs/specs/pike/approved/resource-provider-traits.html
- http://specs.openstack.org/openstack/nova-specs/specs/queens/approved/ironic-driver-traits.html
- http://specs.openstack.org/openstack/nova-specs/specs/queens/approved/request-traits-in-nova.html
- http://specs.openstack.org/openstack/nova-specs/specs/queens/approved/add-trait-support-in-allocation-candidates.html

5.15.176 Nodes tagging

https://bugs.launchpad.net/ironic/+bug/1526266

This aims to add support for tagging nodes.

Problem description

Ironic should have tags field for every node, which can be used to divide the nodes to some groups. then we can do list by tag to get a group of nodes with same properties like hardware specs.

Proposed change

- Add APIs that allows a user to add, remove, and list tags for a node.
- Add tag filter parameter to node list API to allow searching for nodes based on one or more string tags.

Alternatives

Current chassis object is kind of an alternative for grouping nodes.

Data model impact

A new *ironic.objects.tags.NodeTagList* object would be added to the object model.

The *ironic.objects.tags.NodeTagList* field in the python object model will be populated on-demand (i.e. not eager-loaded).

A tag should be defined as a Unicode string no longer than 255 characters in length, with an index on this field.

Tags are strings attached to an entity with the purpose of classification into groups. To simplify requests that specify lists of tags, the comma character is not allowed to be in a tag name.

For the database schema, the following table constructs would suffice

```
CREATE TABLE node_tags (
    node_id INT(11) NOT NULL,
    tag VARCHAR(255) CHARACTER SET utf8 NOT NULL,
```

(continues on next page)

(continued from previous page)

```
    PRIMARY KEY (node_id, tag),
    KEY (tag),
    FOREIGN KEY (node_id)
      REFERENCES nodes(id)
      ON DELETE CASCADE,
)
```

REST API impact

We will follow the API Working Groups specification for tagging, rather than invent our own.

Will support addressing individual tags.

RPC API impact

None

State Machine Impact

None

Client (CLI) impact

Add tags CRUD operations commands:

- ironic node-tag-list <node uuid>
- ironic node-tag-update <node uuid> <op> <tags>

<op> Operation: add or remove

For individual tag: * ironic node-tag-add <node uuid> <tag> * ironic node-tag-remove <node uuid> <tag>

Add tag-list filtering support to node-list command:

- ironic node-list tag tag1 tag tag2
- ironic node-list tag-any tag1 tag-any tag2
- ironic node-list not-tag tag3

Multiple tag will be used to filter results in an AND expression, and tag-any for OR expression, allowing for exclusionary tags via the not-tag option.

Driver API impact

None

Nova driver impact

The tags information can be used for nova but its not being considered as part of this spec, and may be addressed at a later time.

Ramdisk impact

N/A

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

None

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
niu-zglinux

Work Items

- Add *node_tags* table with a migration.
- Add DB API layer for CRUD operations on node tags.
- Added DB API layer for node tag-list filtering support.
- Add NodeTag, NodeTagList objects and a new tags field to Node object.
- Add REST API for CRUD operations on node tags.
- Add REST API for node tag-list filtering support.
- python-ironicclient additions and modifications.

Dependencies

None

Testing

Add unit tests. Add tempest API tests.

Upgrades and Backwards Compatibility

Add a migration script for DB.

Documentation Impact

Ironic API and python-ironicclient will need to be updated to accompany this change.

References

1. http://specs.openstack.org/openstack/api-wg/guidelines/tags.html

5.15.177 Add notification support to ironic

https://bugs.launchpad.net/ironic/+bug/1526408

Near real-time inter-service notifications in OpenStack have a variety of use cases. At the Mitaka design summit, some of the use cases discussed for ironic included integration with billing systems, using notifications to support an operations panel to keep track of potential failures, and integration with TripleO.[1] Nova has supported some form of notifications since the Diablo cycle.[2] This spec proposes a design for a general notification schema thats versioned and structured but extensible enough to support future needs.

Problem description

Ironic doesnt currently support any type of notifications for external services to consume.

This makes it difficult for operators to monitor and keep track of events in ironic like node state changes, node power events, and successful or failed deployments. Sending notifications that contain relevant information about a node at different points in time, like state transitions, would make it easier for operators to use external tools to debug unexpected behavior.

Not having notifications also makes it difficult for external services to identify state changes in ironic. Nova and inspector have to poll ironic repeatedly to see when a state change has occurred. Adding notifications for cases like these will allow ironic to push out these event changes rather than putting the bulk of this burden on external services.

This is especially problematic if ironic were ever to be used in a standalone deployment without an external service like Nova that keeps track of resources and emits its own notifications, since the deployers of ironic wouldnt have any way to keep track of any state changes outside of querying the ironic API periodically or looking at logs.

[1] Summit discussion: https://etherpad.openstack.org/p/summit-mitaka-ironic-notifications-bus
[2] https://blueprints.launchpad.net/nova/+spec/notification-system

Proposed change

Notification support will be added to ironic using the notifier interface from oslo.messaging.[3] oslo.messaging uses the following format for notifications:

```
{'message_id': six.text_type(uuid.uuid4()),
 'publisher_id': 'compute.host1',
 'timestamp': timeutils.utcnow(),
 'priority': 'WARN',
 'event_type': 'compute.create_instance',
 'payload': {'instance_id': 12, ... }}
```

Ironic notifications will adhere to that format.

Base classes will be created to model the notification format. The fields of the notification that arent auto-generated (everything except message_id and timestamp) will be defined as follows for ironic:

- publisher_id will be taken from the service emitting the notification and the hostname
- priority will be one of DEBUG, WARN, INFO, or ERROR, depending on the notification. Whichever level a notification uses should follow the OpenStack logging standards.[4]
- event_type will be a short string describing the notification. Each string will start with baremetal. to distinguish ironic notifications from other notifications on the message bus.

  This will be followed by the object thats being acted on, a descriptor of the action being taken, and the status of the action (start, end, error, or success). These statuses will have the following semantics:

  - start will be used when an action that is not immediate begins.
  - end will be used when an action that is not immediate succeeds.
  - error will be used when an action fails, regardless of whether the action is immediate or not.
  - success will be used when an action that succeeds almost immediately, and there is no need to report the intermediate resource states.

  event_type will have a base class defining these fields, with subclasses for each type of event that occurs. These subclasses will be versioned objects. Each of these fields will be separated by a period in the event_type string sent with the notification.

  Given the above information, event_type will end up as a string on the message bus with the following format: `baremetal.<object>.<action>.<status>`

  A few examples of potential event_type strings:

  - baremetal.node.power_set.start
  - baremetal.node.power_set.end
  - baremetal.node.provision_set.start
  - baremetal.node.provision_set.end
  - baremetal.conductor.init.success
  - baremetal.conductor.stop.success

[3] http://docs.openstack.org/developer/oslo.messaging/notifier.html
[4] https://wiki.openstack.org/wiki/LoggingStandards#Log_level_definitions

- payload will be a versioned object related to the notification topic. This should include relevant information about the event being notified on. For the notifications above about a node, for example, we would send an object containing a limited subset of the fields of the Node object, removing sensitive fields like private credentials as well as fields with an unknown length, along with the version of the notification object. A separate versioned notification object must be created for each type of notification to keep notification versions separate from other object versions in ironic. This is to prevent notifications from automatically changing as new fields are added to other objects like the Node object.

The initial implementation will consist of the base classes needed for modeling the notification format and emitting notifications as well as the power state notifications described above.

Alternatives

One alternative would be to modify oslo.messagings notifier interface to provide the base class defining a notification as a versioned object. The advantage of this is that we wouldnt have to rely on the payload to do versioning. The disadvantage is that it would require significant changes to oslo.messaging which affect other projects and require changes to every notification consumer as well as any downstream tooling built around the existing notification format.

We could also have a global notification version that gets incremented with each change to any notification. This doesnt seem desirable because it could potentially miss changes if a separate ironic object included in the payload like the Node object gets changed and the notification itself doesnt get its version bumped.

Data model impact

No database changes will be required for this feature.

The change will introduce new versioned base objects for the general notification schema and additional subclasses for individual notification types. The base classes will look similar to those in the proposal in the nova versioned notifications spec.[5]

State Machine Impact

No impact on states or transitions themselves.

Notifications will be sent out on node provision state changes if notifications are enabled. This can be achieved by sending notifications every time a TaskManagers process_event method starts and successfully executes.

All information related to a nodes previous, current, and new target state will be included in the notifications. The .start notification will have the current provision_state before the state change and the target_provision_state. After an event is successfully processed, the .end notification will include the current provision_state (the .start notifications target state) and the new target_provision_state. The notifications will also include the name of the event that caused the state change. This will be useful for disambiguating between cases where there are multiple potential transitions from one state to another.

[5] Nova versioned notifications spec: https://github.com/openstack/nova-specs/blob/master/specs/mitaka/approved/versioned-notification-api.rst

REST API impact

None.

Client (CLI) impact

None.

RPC API impact

No impact from an API standpoint.

Modifications to the implementation of certain conductor RPC API methods will need to be made for notifications that are sent when an RPC is dispatched to a worker, however. See Driver API Impact for an example of how this might be done for power notifications.

Driver API impact

No impact from an API standpoint.

Notifications related to power state changes will be added, but that can be done without modifying any of the driver classes in the following manner:

1) Send a baremetal.node.set_power_state.start notification after the ConductorManager receives the change_node_power_state call as a conductor background task.

2) On success, after the dispatched call to node_power_action finishes without raising an exception, send a baremetal.node.set_power_state.end notification.

3) On error, the power_state_error_handler hook will be called in the conductor manager. Send a baremetal.node.set_power_state.error notification here.

Nova driver impact

None.

Ramdisk impact

N/A

Security impact

None.

Other end user impact

None, except a message bus will have to be used if a deployer wants to use the notification system.

Scalability impact

When enabled, notifications will put additional load on whichever message bus the notifications are sent to.

Performance Impact

When enabled, code to send the notification will be called each time an event occurs that triggers a notification. This shouldnt be much of a problem for ironic itself, but the load on whatever message bus is used should be considered (see Scalability Impact).

Other deployer impact

The following configuration options will be added:

- The notification_transport_url option needed by oslo.messaging.[6] Defaults to None which indicates that the same configuration thats used for RPC will be used.
- A notification_level string parameter will be added to indicate the minimum priority level for which notifications will be sent. Available options will be DEBUG, INFO, WARN, ERROR, or None to disable notifications. None will be the default.

  An alternative to the notification_level global config option would be to create specific config options defining whether a particular notification type should be sent. This is what nova does, but summit discussions indicated that consistency is preferable.

Developer impact

Developers should adhere to proper versioning guidelines and use the notification base classes when creating new notifications.

Implementation

Assignee(s)

Primary assignee:

- mariojv

Other contributors:

- lucasagomes

Work Items

- Create notification base classes and tests
- Write documentation for how to use the base classes consistently across all ironic notifications
- Implement an example of a notification for when a node power state is changed

Dependencies

None.

Testing

Unit tests for both the base classes and the node power state notification will be added.

[6] http://docs.openstack.org/developer/oslo.messaging/opts.html

Upgrades and Backwards Compatibility

No impact, but modifications to notifications created in the future must be checked for backwards compatibility.

Documentation Impact

- Developer documentation will be added for how to add new notifications or modify existing notifications
- Document an example of what an emitted notification will look like
- Documentation should be added for each notification indicating when its expected to be emitted

References

5.15.178 Nova-compatible Serial Console

https://bugs.launchpad.net/ironic/+bug/1553083

This implements console interfaces using socat which provides nova- compatible serial console capability.

Problem description

Currently, ironics only console interface is based on shellinabox, which provides a stand-alone web console, and is not compatible with nova-serialproxy.

Proposed change

In order to address the problem of not having a serial console compatible with nova, this spec proposes using a command line tool `socat`[1] in conjunction with IPMI Serial-Over-Lan capabilities. `socat` is a command line based utility that establishes two bidirectional byte streams and transfers data between them. This application allows us to activate the ipmitool Serial-over-LAN process and redirect it through a TCP connection which can then be consumed by the `nova-serialproxy` service.

Each console (socat + ipmitool) will run on its own process on the same host that the ironic-conductor which is currently managing that node is running. `socat` will run first, then it will execute ipmitool when it has connections from `nova-serialproxy`, and will work like a `bridge` between them.

Start/stop of an ironic-conductor process

- When ironic-conductor starts, if console mode of the node is true, start socat also.
- When ironic-conductor stops, if console is started, stop it.
- About takeover work, were planning:
  - When an ironic-conductor stops, console session will be stopped due to security reason. In this case, if there are other ironic-conductors, they takeover nodes and enables their console session again.
  - When ironic-conductor starts, if there are console enabled nodes, the ironic-conductor starts their console.
- Start/stop console will be implemented with subprocess.Popen.

[1] http://linux.die.net/man/1/socat

- About start/stop socat, were planning to implement a new console interface IPMISocatConsole and implement same methods as shellinabox classes. About this, discussed in:[2] .
- About reconnection, for example, in case of temporary network problem with using Horizon, Closed message will be shown. And socat itself supports session reconnect from client side, so that, when the network problem is resolved, users can try to reconnect.

Specify which of shellinabox or socat to use

Were planning to specify which driver to use shellinabox or socat by setting driver like `pxe_ipmitool_socat` or `agent_ipmitool_socat`. (Please see `Other deployer impact` section.)

Alternatives

Creating a new service ironic-xxx instead of adding a new ConsoleInterface to `ironic-conductor` . The upside of new service is that it can be scaled independently, and has no implications on conductor failover. However it will need its own HA model as well, and will be more work for developers (API, DB, driver,).

Data model impact

None

State Machine Impact

None

REST API impact

The response body of GET /v1/nodes/<UUID>/states/console contains a JSON object like below:

```
{
  "console_enabled": true,
  "console_info": {
    "url": <url>,
    "type": <type>
  }
}
```

In case of using socat instead of shellinabox, <type> will be socat and <url> is like tcp://<host>:<port>.

Client (CLI) impact

None

RPC API impact

None

[2] https://review.opendev.org/#/c/293873/

Driver API impact

None

Nova driver impact

get_serial_console() will be implemented in ironic driver of Nova. It returns a dictionary, similar to nova.virt.libvirt.driver.LibvirtDriver.get_serial_console(). No other impact for nova, and nova-serialproxy works well with the new one. And also, nova has agreed to the nova side of the work[3].

Ramdisk impact

None

Security impact

The connection between nova-serialproxy and socat is TCP based, like KVM. Socat supports OpenSSL connections, so we can improve the security in the future.

Other end user impact

None

Scalability impact

If a conductor can service 1000 nodes, and a process is created for a console to each node, but its the same scalability issue as shellinabox.

Performance Impact

None

Other deployer impact

To use socat serial console, deployer needs to specify new driver. For example, to use PXE + IPMItool + socat, specify `pxe_ipmitool_socat`. To use IPA + IPMItool + socat, specify `agent_ipmitool_socat`. To use existing shellinabox console, deployer doesnt need to change anything. The new console interface `IPMISocatConsole` will be supported by two new drivers: `pxe_ipmitool_socat` and `agent_ipmitool_socat`. After `Driver composition reform`[4] is implemented, this feature will be available for a lot more drivers (or hardware types).

About configuration options, existing options `terminal_pid_dir`, `subprocess_checking_interval`, `subprocess_timeout` are available for socat in the same way as shellinabox. `terminal_cert_dir` is not used in the case of socat because SSL is not supported. `terminal` is not used in the case of socat because hard-coded `socat` is used in the code, and absolute path is not needed because its distro specific, in Ubuntu for example its `/usr/bin/socat`, but it might be different in other distros.

[3] https://blueprints.launchpad.net/nova/+spec/ironic-serial-console-support
[4] https://review.opendev.org/#/c/188370/

Developer impact

None

Implementation

Assignee(s)

Primary assignee:

- Akira Yoshiyama <akirayoshiyama@gmail.com>

Other contributors:

- Dao Cong Tien <tiendc@vn.fujitsu.com>
- Nguyen Tuong Thanh <thanhnt@vn.fujitsu.com>
- Cao Xuan Hoang<hoangcx@vn.fujitsu.com >
- Hironori Shiina <shiina.hironori@jp.fujitsu.com>
- Yuiko Takada Mori <y-mori@ti.jp.nec.com>

Work Items

- Implement `IPMISocatConsole` and `NativeIPMISocatConsole` class inherited from `base.ConsoleInterface`.

Dependencies

None

Testing

Unit Testing will be added.

Upgrades and Backwards Compatibility

None

Documentation Impact

Add configuration description to the install guide.

References

5.15.179 Dynamic allocation of nodes for the OneView drivers

https://bugs.launchpad.net/ironic/+bug/1541096

This spec proposes a change in the way the OneView drivers allocate nodes. The new model will allocate resources in OneView only at boot time, avoiding that idle hardware in Ironic is still blocked from OneViews users perspective.

In this spec, the following terms will be used to refer to OneView resources:

- Server Hardware (SH): corresponds to a physical server. A Server Hardware can have only one Server Profile applied at a time.

- Server Profile (SP): representation of hardware configuration, such as network configuration, boot settings, firmware version, as well as features configurable through BIOS.
- Server Profile Template (SPT): template used to define a reference configuration of a set of Server Profiles.

Problem description

The current version of the OneView drivers consider what we call *pre-allocation* of nodes. This means that when a node is registered in Ironic, there must be a Server Profile applied to the Server Hardware represented by the given node.

In OneView, when a Server Hardware has a Server Profile applied to it, the understanding is that such hardware is being used.

The problem with *pre-allocation* is that even when a node is *available* in Ironic and, therefore, there is no instance associated to it, the resource in OneView is considered to be allocated. This means that no other OneView user can use and control this resource.

Therefore, in an environment shared by Ironic and OneView users, to simply allocate a pool of Server Hardware items to Ironic, for which we do not know when they are going to be used, will result in hardware reserved but not in use, which from many perspectives, specially for OneView users, is undesirable.

The ability to dynamically allocate nodes is currently missing from the set of OneView drivers. By adding this feature, the drivers will be able to have OneView resources allocated to Ironic only at boot time. Thus, this will enable the hardware pool to be actually shared among OneView and Ironic users.

Proposed change

In order to support dynamic allocation of nodes, some steps in the process of validating and deploying an OneView node need to be changed.

A node will be considered to be free for Ironic when (1) there is no Server Profile applied to the Server Hardware the node represents, or (2) there is a Server Profile applied, but this Server Profile is consistent with what is registered in Ironics node data structure. This means that the Server Profiles URI is equal to the value of the field *applied_server_profile_uri* in the *driver_info* namespace of the given node in Ironic.

Therefore, the following rules define the ownership of the node:

- SH without server_profile - Free
- SH.server_profile == node.driver_info.applied_server_profile_uri - Used by Ironic
- SH.server_profile != node.driver_info.applied_server_profile_uri - Used by OneView

When following this approach, a Server Profile is no longer required to validate a node. The *validate* methods of the *Power* and *Management* interfaces of the driver must be changed.

In the proposed model, if a node is not free for Ironic (i.e., it is being used by OneView) the validation will fail. However, as we expect nodes that are used by OneView to be moved to *manageable* and be cleaned before being made available again in Ironic, if the *target_provision_state* of a node is *manageable*, then this check will be skipped.

A node free for Ironic can be claimed by a OneView user at any moment. In that case, the node would still appear to be *available* in Ironic. To settle this case, a driver periodic task will be issued to detect nodes in use by OneView users and set them to *manageable* state and put into maintenance mode with a proper maintenance reason message. As soon as the node is freed by the OneView user, another driver periodic

task will remove the maintenance mode and *provide* the node back into available. This is necessary to ensure the node is cleaned before being provided to an Ironic user.

If such node is scheduled for deployment prior to the execution of the periodic task, it will go to the *deploy failed* state and enter the cleaning process, that should fail since the node is in use by a OneView user. Nova scheduler will be able to pick a different node. A third periodic task will then be responsible to detect such failures and move the node back to *manageable* state and set maintenance mode until freed.

Some changes in the cleaning process must also be considered. In order to clean a node, a Server Profile needs to be assigned to it. Considering that, if a node has no Server Profile applied to it, a new temporary one will be created based on the Server Profile Template of this node, and applied to such hardware for cleaning purposes. If there is a Server Profile already applied to the node, it will be reused. After the cleaning is complete, in both cases, such Server Profile will be removed.

From a technical perspective, the *iscsi_pxe_oneview* and *agent_pxe_oneview* drivers will now implement:

- oneview.deploy.OneViewIscsiDeploy
- oneview.deploy.OneViewAgentDeploy

Both interfaces will override three methods:

- *prepare_cleaning*:

If the node is in use by OneView, an exception will be raised and the node goes to the *clean failed* state. Otherwise, cleaning steps will be performed and additional actions taken according to the following:

- **If there is no Server Profile assigned to the nodes Server Hardware:**
  - Server Profile will be created according to the Server Profile Template of the node;
  - Such Server Profile will be applied to the Server Hardware the node represents;
  - *applied_server_profile_uri* field will be added to the *driver_info* namespace of the node;
- *tear_down_cleaning*:

If the node is in use by Ironic, the following actions will be taken:

- *applied_server_profile_uri* field will be deleted from the *driver_info* namespace of the node;
- Server Profile will be removed from the Server Hardware the node represents;
- *prepare*:

If the node is in use by OneView, an exception will be raised and the node goes to a *deploy failed* state. Otherwise, additional actions will be taken as follows:

- Server Profile is applied to the Server Hardware represented by the node. The information required to perform such task will be recovered from the Server Profile Template indicated in the *server_profile_template_uri* inside the *properties/capabilities* namespace of the node.
- *applied_server_profile_uri* field will be added to the *driver_info* namespace of the node;

The interfaces will also implement three new driver periodic tasks:

- *_periodic_check_nodes_taken_by_oneview*:

This driver periodic task will check for nodes that were taken by OneView users while the node is in available state and set the node to maintenance mode with an appropriate maintenance reason message and move the node to *manageable* state.

- *_periodic_check_nodes_freed_by_oneview*:

This driver periodic task will be responsible to poll the nodes that are in maintenance mode and on *manageable* state to check if the Server Profile was removed, indicating that the node was freed by the OneView user. If so, itll *provide* the node, that will pass through the cleaning process and become available to be provisioned.

- *_periodic_check_nodes_taken_on_cleanfail*

This last driver periodic task will take care of nodes that would be caught on a race condition between OneView and a deploy by Ironic. In such cases, the validation will fail, throwing the node on *deploy fail* and, afterwards on *clean fail*. This task will set the node to maintenance mode with a proper reason message and move it to *manageable* state, from where the second task can rescue the node as soon as the Server Profile is removed.

A new configuration will be created on *[oneview]* section to allow operators to manage the interval in which the periodic tasks will run:

```
[oneview]
...
periodic_check_interval=300
```

Alternatives

Today, there is no other way to enable dynamic allocation of nodes with the OneView drivers.

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

ironic CLI

None

openstack baremetal CLI

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

N/A

Security impact

When a machine previously in use by an OneView user is released, its disks are not erased in the process since OneView does not perform such cleaning tasks. This means that once a node is released, some remaining data from its previous user can be available for Ironic. To prevent this leftover data from being exposed to Ironic users, the driver will move these machines to *manageable* state through its periodic tasks, where Ironic will require them to go through cleaning before being made *available* again. Note that if the cleaning feature is disabled in Ironic, OneView users are responsible for manually erase such disks prior to releasing the hardware.

Other end user impact

None

Scalability impact

None

Performance Impact

In most cases, applying a Server Profile in OneView takes less than 2 minutes.

In the few cases it can take longer, e.g. in firmware upgrades, the user must configure timeouts accordingly. Documentation will be provided on how to do it.

The performance impact of the periodic tasks on Ironic conductor will be proportional to the number of nodes being managed since theyll poll nodes in OneView to check if nodes were taken/returned by OneView users. Well minimize that impact by polling only nodes on specific known states as said previously on this spec. Yet, the *periodic_check_interval* can be adjusted according to OneView usage behavior and the number of nodes enrolled to reach an optimized performance of the conductor.

Other deployer impact

After this change is merged, the way the OneView drivers allocate nodes will be different. Deployers should be aware that:

- For existing nodes hosting instances, the *applied_server_profile_uri* field must be added to the *driver_info* namespace of the node.
- Nodes that had a Server Profile assigned to them but were not actually in use (according to the *pre-allocation* model), can have their Server Profiles removed and still be available in Ironic.

In order to ease this process, a migration tool will be provided. Check *Upgrades and Backwards Compatibility* for more details.

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
liliars

Other contributors:
sinval thiagop gabriel-bezerra marcusrafael nicodemos caiobo

Work Items

- Implement oneview.deploy.OneViewIscsiDeploy
- Implement oneview.deploy.OneViewAgentDeploy
- Override *prepare*, *prepare_cleaning*, *tear_down_cleaning* methods for both interfaces
- Implement the driver periodic tasks to deal with nodes taken by OneView users
- Write tests for such scenarios
- Document these changes

Dependencies

- python-oneviewclient version will be bumped

Testing

- Unit-tests will be implemented for the changes;
- The OneView third party CI will be used to provide a suitable test environment for tests involving a OneView appliance and specific hardware;

Upgrades and Backwards Compatibility

As said in *Other deployer impact* section of this spec, to migrate nodes that are in use with the *pre-allocation* model, one should add the field *applied_server_profile_uri* to the *driver_info* namespace of the node. Nodes that are on *available* state needs to have their Server Profiles removed. To ease this upgrade process on running deployments of Ironic, a migration tool will be provided and properly referenced in the drivers documentation.

Operators might have to increase the value of *workers_pool_size* and *periodic_max_workers* settings to allow to increase the greethread pool size and allow more than one parallel task to deal with nodes taken/returned on each periodic task as the pool of nodes on OneView increases.

Deprecation policy for *pre-allocation*:

- Newton:
  - Both pre-allocation and dynamic allocation will be supported

  - Flag to indicate whether dynamic allocation is enabled in driver_info
  - Driver defaults to pre-allocation in case the flag is missing
  - Script provided to ease migration process.
  - Deprecate *pre-allocation* feature in the driver code.
- Ocata:
  - Both pre-allocation and dynamic allocation will continue to be supported (due to deprecation process)
- P:
  - Drop support of *pre-allocation*
  - Flag ignored.

Documentation Impact

OneView drivers documentation will be updated accordingly. Some topics to be addressed are as follow:

- Definition of the new dynamic allocation model;
- Addition of fields in the node;
- Information regarding the migration process from the *pre-allocation* model;
- New configuration options and possible improvements;

References

OneView page
http://www8.hp.com/ie/en/business-solutions/converged-systems/oneview.html

OneView 2.0 REST API Reference
http://h17007.www1.hp.com/docs/enterprise/servers/oneview2.0/cic-rest/en/content/

python-oneviewclient
https://pypi.org/project/python-oneviewclient

5.15.180 OSC commands to get descriptions of driver-related information

- https://bugs.launchpad.net/python-ironicclient/+bug/1619052
- https://bugs.launchpad.net/python-ironicclient/+bug/1619053

Almost all the ironic CLI commands have corresponding OpenStack Client (OSC) commands. This was done in Implementation of ironic commands as an OSC plugin.

There are two ironic CLI commands:

- `ironic driver-properties` and
- `ironic driver-raid-logical-disk-properties`

that do not have corresponding OSC commands. This specification proposes OSC commands for them.

Properties of hardware types with non-default interfaces is addressed in RFE to get properties for a dynamic driver and non-default interfaces. That proposal will most likely result in a richer REST API (and corresponding OSC commands). In light of that, we propose simple OSC commands with the goal of providing equivalent behaviours to the ironic CLI commands, and nothing more.

Problem description

This table shows the ironic driver-related commands and their associated OSC commands.

ironic	openstack baremetal
driver-get-vendor-passthru-methods <driver>	driver passthru list <driver>
driver-list	driver list
driver-show <driver>	driver show <driver>
driver-vendor-passthru <driver> <method>	driver passthru call <driver> <method>
driver-properties <driver>	?
driver-raid-logical-disk-properties <driver>	?

The question is what to use for the OSC commands corresponding to these ironic commands:

- `ironic driver-properties <driver>`
- `ironic driver-raid-logical-disk-properties <driver>`.

These commands return tables with two columns; the names and descriptions for driver-related information.

To complicate things a bit, there is a lack of symmetry. Although the names (and descriptions) are available via the driver, the values for these are specified via the nodes that use these drivers, rather than directly via the driver.

There have been a few discussions related to this:

- http://lists.openstack.org/pipermail/openstack-dev/2016-September/103490.html
- briefly discussed at the OpenStack PTG in Atlanta in February 2017, see How to call (a name for it) the OSC command for listing RAID properties? (dtantsur) in the PTG operations etherpad.

Properties of hardware types with non-default interfaces is addressed in RFE to get properties for a dynamic driver and non-default interfaces. That proposal will most likely result in a richer REST API (and corresponding OSC commands). In light of that, we propose simple OSC commands with the goal of providing equivalent behaviours to the ironic CLI commands, and nothing more.

ironic driver-properties <driver>

The `ironic driver-properties <driver>` command returns the properties of a driver that must be specified via the nodes that use that driver.

For example, `ironic driver-properties agent_ipmitool` returns a table of information:

```
+--------------------------+-----------------------------------------------+
| Property                 | Description                                   |
+--------------------------+-----------------------------------------------+
| deploy_forces_oob_reboot | Whether Ironic should force a reboot of t...|
| deploy_kernel            | UUID (from Glance) of the deployment kern...|
| deploy_ramdisk           | UUID (from Glance) of the ramdisk that is...|
| image_http_proxy         | URL of a proxy server for HTTP connection...|
| image_https_proxy        | URL of a proxy server for HTTPS connectio...|
| image_no_proxy           | A comma-separated list of host names, IP ...|
| ...                      | ...                                           |
+--------------------------+-----------------------------------------------+
```

The values for these driver properties are set per node, for each node that uses the driver. The information is in the nodes `driver_info` dictionary field. They can be set when creating a node or when updating a node, for example with `openstack baremetal node set <node> --driver-info`.

ironic driver-raid-logical-disk-properties <driver>

The `ironic driver-raid-logical-disk-properties <driver>` command returns the RAID logical disk properties that can be specified for a particular driver.

For example, `ironic driver-raid-logical-disk-properties agent_ipmitool` returns a table of information:

```
+--------------------------+---------------------------------------------+
| Property                 | Description                                 |
+--------------------------+---------------------------------------------+
| controller               | Controller to use for this logical disk. ...|
| disk_type                | The type of disk preferred. Valid values ...|
| interface_type           | The interface type of disk. Valid values ...|
| is_root_volume           | Specifies whether this disk is a root vol...|
| number_of_physical_disks | Number of physical disks to use for this ...|
| physical_disks           | The physical disks to use for this logica...|
| raid_level               | RAID level for the logical disk. Valid va...|
| share_physical_disks     | Specifies whether other logical disks can...|
| size_gb                  | Size in GiB (Integer) for the logical dis...|
| ...                      | ...                                         |
+--------------------------+---------------------------------------------+
```

The values for these disk properties are set per node, for each node that uses the driver and RAID. This information is in the nodes `target_raid_config` field and can be set (in JSON format) via the nodes set raid config API, or via `openstack baremetal node set <node> --target-raid-config`.

Proposed change

The OpenStackClient command structure is:

```
openstack [<global-options>] <object-1> <action> [<object-2>] [<command-
↪arguments>]
```

ironic driver-properties <driver>

The <object-1> part will be baremetal driver property.

For the <action>, there were discussions at the PTG about the merits of using list versus show. In the context of existing OSC commands using these two action words, neither of these action words seems to fit with the two commands, since they return descriptions (or documentation) on what is available. However, since this will very likely be replaced by a richer set of OSC commands (resulting from RFE to get properties for a dynamic driver and non-default interfaces), we keep it simple and use list. (Using list might imply that a corresponding show exists to drill down and get more information. The converse is true too though; using show might imply a corresponding list command.)

The OSC command is:

```
openstack baremetal driver property list <driver>
```

For example:

```
$ openstack baremetal driver property list agent_ipmitool

+-------------------------+-----------------------------------------------+
| Property                | Description                                   |
+-------------------------+-----------------------------------------------+
| deploy_forces_oob_reboot | Whether Ironic should force a reboot of t...|
| deploy_kernel           | UUID (from Glance) of the deployment kern...|
| deploy_ramdisk          | UUID (from Glance) of the ramdisk that is...|
| image_http_proxy        | URL of a proxy server for HTTP connection...|
| image_https_proxy       | URL of a proxy server for HTTPS connectio...|
| image_no_proxy          | A comma-separated list of host names, IP ...|
| ...                     | ...                                           |
+-------------------------+-----------------------------------------------+
```

ironic driver-raid-logical-disk-properties <driver>

Again, since this will very likely be replaced by a richer set of OSC commands (that will result from RFE to get properties for a dynamic driver and non-default interfaces), we propose a simple OSC command.

<object-1> would be baremetal driver raid property and <action> would be list:

```
openstack baremetal driver raid property list <driver>
```

Alternatives

There are alternatives, but in the interest of keeping this simple and deprecating it when RFE to get properties for a dynamic driver and non-default interfaces is available, we are focussed here on only providing OSC-equivalent commands that may not be flexible or extensible, but provide equivalence to the ironic CLI ones.

Data model impact

None.

State Machine Impact

None.

REST API impact

None.

Client (CLI) impact

This specification is about the OSC CLI.

ironic CLI

None.

openstack baremetal CLI

See above.

RPC API impact

None.

Driver API impact

None.

Nova driver impact

None.

Ramdisk impact

None.

Security impact

None.

Other end user impact

None.

Scalability impact

None.

Performance Impact

None.

Other deployer impact

None.

Developer impact

None.

Implementation

Assignee(s)

Primary assignee: rloo (Ruby Loo)

Other contributors: galyna (Galyna Zholtkevych)

Work Items

- Code the two OSC commands in python-ironicclient.

Dependencies

None.

Testing

Unit and functional testing similar to what exists for other OSC commands.

Upgrades and Backwards Compatibility

None.

Documentation Impact

None. ironic doesnt have OSC-command related documentation.

References

- Implementation of ironic commands as an OSC plugin
- OpenStackClient command
- PTG operations etherpad
- updating a node
- nodes set raid config
- http://lists.openstack.org/pipermail/openstack-dev/2016-September/103490.html
- RFE to get properties for a dynamic driver and non-default interfaces

5.15.181 Override PXE options via Glance property

https://bugs.launchpad.net/ironic/+bug/1526409

The proposal presents the work required to allow PXE boot an image using a specific kernel command line for that image.

Problem description

Some images requires having specific kernel command line to boot correctly. Currently, Ironic tries to determine things like the root filesystem the image is on (root= kernel command line) by getting the UUID of the image filesystem and setting as the root= parameter. This is not correct for all images because some may need to boot from a different root filesystem, e.g:

- The device path of its LVM volume [root=/dev/mapper/vg-lv_root]

- From one squashfs filesystem [root=live:<path>]
- A btrfs subvolume [root=/dev/disk/by-uuid/<UUID of btrfs-root>]

As a real example, the Fedora Atomic uses `lvm` and `ostree`. It requires a kernel command line as the example below to boot properly:

```
nofb nomodeset vga=normal console=tty1 no_timer_check
rd.lvm.lv=atomicos/root root=/dev/mapper/atomicos-root
ostree=/ostree/boot.0/fedora-atomic/a002a2c2e44240db614e09e82c/0
```

Proposed change

In the same way Ironic look at the Glance image properties to find out the Glance UUID for the images kernel and ramdisk (kernel_id and ramdisk_id) and populate the nodes `instance_info` field with the `kernel` and `ramdisk` keys.

This spec propose having a new images property field called `kernel_cmdline` that Ironic will look at, and if present, it will populate the nodes `instance_info` field with a `kernel_cmdline` key.

If the `instance_info.kernel_cmdline` is populate Ironic will skip getting the root partition filesystem UUID and will use the command line from the `instance_info` field instead of the default one in the template. Its important to note that the values present at the `pxe_append_params` configuration option will still be appended at the end of the images custom kernel command line by Ironic.

Setting the keys in the `instance_info` field is important because it also allows Ironic to operate in standalone mode without Glance.

Alternatives

Always use whole disk images when deploying images that requires a specific kernel command line

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

N/A

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

Deployers that wants to deploy an image with a specific kernel command line should know and set it in Glance images property prior to trying to boot the image.

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
lucasagomes <lucasagomes@gmail.com>

Other contributors:
None

Work Items

- Make Ironic look at the `kernel_cmdline` image property in Glance and if present set it to the nodes `instance_info` field

- When preparing to boot the users image, make Ironic check if the nodes `instance_info` field contains a key called `kernel_cmdline` (along with `kernel_id` and `ramdisk_id`) and if so, use that kernel command line to boot the image.

Dependencies

None

Testing

- Unit Tests

Upgrades and Backwards Compatibility

None

Documentation Impact

The Ironic deploy documentation will be updated to reflect the changes made by this spec.

References

- Kernel parameters
- Project atomic

5.15.182 Ownership Information Storage

https://storyboard.openstack.org/#!/story/2001814

In many large businesses with hardware fleets, there may be a variety of ownership of the underlying hardware for tracking purposes. A good example of this is in a hybrid service provider scenario where an operator may directly own a portion of the hardware, may have the hardware on lease, and may ultimately have customer owned hardware.

We simply cannot model this with existing node fields, since there may be resource sharing agreements also in place between the owners of the hardware. And as such, we need some way to store and represent the end owner of the hardware for tracking and logistical purposes.

Problem description

While scenarios differ, ultimately there is a need to be able to store information in a top level fashion about who owns a given piece of hardware.

Information of this sort can be vital when it comes time for tax asset inventories, or just simple tracking of where the hardware came from.

Providing the ability to search and return the hardware that is known to be owned by a particular group allows for faster correlation of the disposition of the hardware for auditing and accounting purposes.

Proposed change

Proposing to add a new informational field to the node object that can be queried via the REST API, and stored in the database. No other initial use of this field is expected.

Future use could also be automatic in the scenario if there is a driver that is aggregating a number of management systems, however that is out of scope.

Alternatives

An operator could potentially store this information in extra, however then they would still need to dump all of the nodes out to obtain a list of nodes with the specific information that is needed. The operator would begin to hit limits with the number of responses from the API, and would need to likely create their own tooling around list processing.

Data model impact

A `owner` field would be added to the nodes table as a `VARCHAR(255)` and will have a default value of `null` in the database.

State Machine Impact

None

REST API impact

An `owner` field will be returned as part of the node. The API shall be updated to allow a user to set and unset the value via the API.

Additionally the GET syntax for the nodes list will be updated to allow a list of matching nodes to be returned.

POST/PATCH operations to the field will be guarded by a new microversion. The field shall not be returned unless the sufficient microversion is supplied for GET operations.

Client (CLI) impact

ironic CLI

None

openstack baremetal CLI

A corresponding change will be necessary to enable the ability for a user to set/unset the value from a command line.

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

None

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

None

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
TheJulia - juliaashleykreger@gmail.com

Other contributors:
None

Work Items

- Add database field.
- Add object field.
- Add REST API functionality and microversion.
- Update REST API documentation.
- Update python-ironicclient.

Dependencies

None

Testing

Basic API CRUD testing will be added. There is no need for additional testing as this is an informational field for the API user/baremetal operator.

Upgrades and Backwards Compatibility

Field will be created as part of the upgrade process with a default value in the database schema.

Documentation Impact

REST API documentation will need to be updated.

References

None

5.15.183 Partition image support for agent driver

https://blueprints.launchpad.net/ironic/+spec/partition-image-support-for-agent-driver

This blueprint suggests to enhance agent driver to support deploy partition images with subsequent boot to happen over pxe or vmedia as specified by the driver.

Problem description

As of now agent driver support only whole disk images that can be deployed on the baremetal. With disk image based deploy, the subsequent boot will happen from the local hard drive. Ironic does not have control over the subsequent boots of the provisioned baremetal node.

Proposed change

- Agent driver validate the specified image type is of partition image(raw) by looking for kernel_id and ramdisk_id image properties.
- Send partition information(root, swap, etc) to the agent ramdisk through image_info.
- Upon receiving the partition information, agent ramdisk will work on the given os_install disk and copy the partition image in the root partition.
- Agent ramdisk sends back the root_uuid to the agent driver on the controller.
- Post deploy, agent driver prepares the config for subsequent boot, either using pxe or vmedia as defined by the driver. Both agent_ipmitool and agent_ilo driver should support deploy with partition images.
- Factor out the partitioning code from ironic into a different library and use it in both IPA and ironic code base.

Alternatives

We can use iscsi method to write partition image on to the target disk. We need agent ramdisk to support iscsi, similar to the ironic DIB element.

Data model impact

None.

State Machine Impact

None.

REST API impact

None.

Client (CLI) impact

None.

RPC API impact

None.

Driver API impact

None.

Nova driver impact

None.

Ramdisk impact

N/A

Security impact

None.

Other end user impact

Ability to deploy partition images on nodes managed by the agent.

Scalability impact

None.

Performance Impact

None.

Other deployer impact

None.

Developer impact

None.

Implementation

Assignee(s)

Primary assignee:
faizan-barmawer

Other contributors:
None

Work Items

- Factor out the partitioning code from ironic into a common library for both IPA and ironic code base.
  1. Move the disk partition code from ironic/common to an oslo incubator project as oslo.libironic
     - ironic/common/disk_partitioner.py
     - Some common disk related functions from ironic/drivers/modules/deploy_utils.py
     - Related test cases.
  2. Use oslo.libironic in IPA
- Make necessary changes in agent driver common code, such as validate, deploy, etc.
- Make necessary changes in agent_ipmitool driver to generate correct pxe config for subsequent reboot.
- Make necessary changes in agent_ilo driver to generate iso for subsequent reboot.
- Make changes in IPA (agent ramdisk) to recognize the incoming image information and take appropriate action to deploy partition image on the disk.

Dependencies

Testing

- Unit testing with partition images with agent_ilo and agent_ipmitool drivers.
- Add specific agent driver test cases with partition images in tempest/devstack.

Upgrades and Backwards Compatibility

Documentation Impact

- Make changes to ironic install guide.

References

5.15.184 Physical Network Awareness

https://bugs.launchpad.net/ironic/+bug/1666009

Ironic ports and portgroups correspond to the physical NICs that are available on baremetal nodes. Neutron ports are logical ports attached to tenant and provider networks. When booting baremetal nodes in a system with multiple physical networks, the mapping between logical and physical ports must account for the physical network connectivity of the system.

This feature aims to make ironic aware of the `physical_network` attribute of neutron networks and network segments.

Problem description

Physical Networks

A neutron port on a provider network or network segment is associated with a physical network. The physical network concept is used to specify the physical connectivity of networks and network segments in the system.

There are many reasons why an operator might use multiple physical networks, including:

- routed provider networks[1]
- security, through physical segregation of traffic
- redundancy, through independence of multiple networks
- traffic isolation (e.g. storage vs. application)
- different characteristics (bandwidth, latency, reliability)
- different technologies (Ethernet, Infiniband, Omnipath, etc.)

It is possible that some or all nodes may be connected to more than one physical network. OpenStack adoption is growing rapidly in the scientific computing community, an area in which the use of nodes connected to multiple physical networks is prevalent.

For context, we provide some examples of how the physical network attribute is used in other OpenStack components. The neutron ML2 Open vSwitch agent is made physical network-aware via the `[OVS] bridge_mappings` configuration option. This option maps physical network names to Open vSwitch bridges that have a physical network interface as a port. This option is used for flat and VLAN network types, and is taken into consideration when binding ports to network segments in the neutron server ML2 driver. This acts both to ensure that physical connectivity allows for the requested logical topology, and also supports hosts being connected to multiple physical networks.

A similar mapping exists in nova for pass-through of PCI physical devices or SR-IOV virtual functions via the `[DEFAULT] pci_passthrough_whitelist` configuration option.

Physical Networks in Ironic

An ironic port is connected to a physical network. As portgroups are a layer-2 construct, all ports in a portgroup should be in the same physical network. If a neutron port is mapped to a port or portgroup and is attached to a neutron network or network segment on a different physical network, there will be no

[1] Neutron routed networks

connectivity between the bare metal nodes NIC and other neutron ports on the network. This is perhaps most obvious when it results in a failure to acquire a DHCP lease for the interface.

The mapping between logical ports in neutron and physical ports and portgroups in ironic has always been somewhat unpredictable[2]. The ironic-neutron integration work added support for local link information for ports[3]. In the interface attach/detach API work[4] ironic moved the responsibility for attaching virtual interfaces from nova to ironic. In both of these features physical network-awareness was seen as out of scope.

Currently when a virtual interface is attached to a node, the procedure used to map it to an ironic port or portgroup is roughly as follows:

- if there is a free portgroup, select one
- else if there is a free port with PXE enabled, select one
- else if there is a free port with PXE disabled, select one
- else fail

This algorithm takes no account of the physical network to which the ports and portgroups are connected, and consequently can result in invalid mappings. Further, there is currently no standard way in ironic to specify the physical network to which a port or portgroup is connected.

Provisioning and Cleaning Networks

The ironic pluggable network provider[5] work added support for attaching nodes to dedicated provisioning and cleaning networks during provisioning and cleaning operations respectively. Currently, all ironic ports and portgroups with PXE enabled are attached to the provisioning or cleaning network. While this ensures that the node has a port on the provisioning or cleaning physical network, it may result in unnecessary neutron ports being created if some of the ironic ports or portgroups are connected to a different physical network. In practice this can be avoided by disabling PXE on ports where it is not required.

Scheduling

In a system with multiple physical networks where not all nodes are connected to every physical network, it becomes possible for a user to request a logical network topology that cannot be realised. Without awareness of the physical networks that each ironic node is connected to, nova cannot reliably schedule instances to ironic nodes. This problem is considered beyond the scope of this spec.

Proposed change

There are four parts to the proposed solution to this problem.

1. It must be possible to specify the physical network to which an ironic port is connected.
2. The ironic network interfaces must account for the physical network of an ironic nodes ports and portgroups when attaching tenant virtual interfaces.
3. When attaching a node to a provisioning or cleaning network, neutron ports should be created only for ironic ports and portgroups on the same physical network as the provisioning or cleaning network.

[2] ports cannot be mapped to networks
[3] Ironic neutron integration
[4] interface attach/detach API
[5] pluggable network providers

4. The binding profiles of attached neutron ports should be updated with the physical network of the ironic port or portgroup to which the port was mapped.

Tagging Ports

There are a few options for how ports might be tagged with a physical network:

1. a new attribute of the `local_link_connection` field
2. a new attribute of the `extra` field
3. a new first class field

Reusing an existing field would certainly be easier to implement than adding a new one, but OpenStack history has frequently shown that buried fields often end up needing to become first class citizens. A relevant example here is the `provider:physical_network` extension field on neutron networks, which later became a first class field on network segments[Page 1360, 1]. Further, if ironic intends to support physical network-aware scheduling in future, the ability to efficiently filter ports by their physical network may be advantageous. This spec therefore proposes to add a new first class `physical_network` field to ironics `Port` object. For backwards compatibility, this field will be optional.

The process of mapping physical networks to ironic ports is out of scope for ironic. This could be done either through a manual procedure or through an automated process using information gathered during a hardware introspection process. For example, if using ironic inspector to perform introspection it would be possible to create an introspection plugin[6] that maps switch IDs discovered via LLDP to physical networks.

Portgroups

The physical network of a portgroup will be determined through the physical network of its constituent ports. All ports in the portgroup must have the same physical network, and this will be enforced in the ironic API when creating and updating ports.

This has the unfortunate consequence of making it rather unwieldy to update the physical network of the ports in a portgroup, since the ports must be removed from the portgroup while their physical network is updated. This may be improved upon in future through the use of a virtual physical network field in the portgroups API that allows simultaneous update of the physical network field of all the ports in the group.

Mapping Logical Ports to Physical Ports

In order to account for physical network connectivity, the virtual interface attachment algorithm must determine the physical networks that the neutron port being attached can be bound to. This information is available via the neutron API as the `physical_network` field on network segments in the ports network or as `provider:physical_network` on the ports network.

The virtual interface attachment mapping algorithm will be modified to the use the following set of criteria listed in order of descending priority:

1. reject ports and portgroups with a non-null physical network that is different than all of the networks physical networks
2. prefer ports and portgroups with a non-null physical network to ports with a null physical network
3. prefer portgroups to ports

[6] introspection plugins

4. prefer ports with PXE enabled to ports with PXE disabled

This algorithm provides backwards compatibility for environments in which the port(s) and/or portgroup(s) associated with the ironic node do not have a `physical_network` property configured.

Provisioning and Cleaning Networks

In ironic network drivers that support network flipping for provisioning and cleaning operations, we will create neutron ports only for those ironic ports and portgroups that have PXE enabled and are on the same physical network as the provisioning or cleaning network in question, or do not have a physical network specified.

Neutron Port Binding Profiles

When attaching virtual interfaces to physical or virtual functions of PCI network devices, nova sets a `physical_network` attribute in the `binding:profile` field of the neutron port. Further research is required to determine what effect it would have if ironic were to do the same.

Alternatives

We could continue to use an unpredictable mapping between logical ports and physical ports. This limits the use of ironic to environments in which there is only one physical network.

We could continue with the existing mapping algorithm in ironic but provide neutron with the information required to determine whether a mapping is valid from the `local_link_connection` binding information. Ironic would then be modified to retry interface attachment with a different neutron port if neutron determined the mapping to be invalid. This method would be inefficient due to the retries necessary.

We could avoid the need to tag ironic ports with a physical network by providing a mechanism to map from the information in their `local_link_connection` fields to a physical network. This would require either an addition to ironics data model to support Switch objects or a new neutron API providing a lookup from switch ID to physical network.

Data model impact

A new `physical_network` field will be added to Port object. In neutron the `Segment` objects `physical_network` field is defined as `sqlalchemy.String(64)`, so the same will be used in ironic.

State Machine Impact

None

REST API impact

The port REST API will be modified to support the new `physical_network` field. The field will be readable by users with the baremetal observer role and writable by users with the baremetal admin role. If the port is a member of a portgroup, the API will enforce that all ports in the portgroup have the same value in their physical network field.

Updates to the physical network field of ports will be restricted in the same way as for other connectivity related fields (link local connection, etc.) - they will be restricted to nodes in the `enroll`, `inspecting` and `manageable` states.

The API microversion will be bumped.

Client (CLI) impact

ironic CLI

The ironic CLI will not be updated.

openstack baremetal CLI

The openstack baremetal CLI will be updated to support getting and setting the `physical_network` field on ports.

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

None

Security impact

This change should increase the potential security level of an ironic bare metal cloud by supporting multiple segregated physical networks and honoring the physical network restrictions assigned by the operator.

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

In order to make use of this feature, deployers must tag ironic ports and portgroups with the physical network to which they are attached. This implies that they must have a mechanism to correctly determine this information.

Developer impact

None

Implementation

Assignee(s)

mgoddard

Work Items

- Modify the ironic port model to include a `physical_network` field.
- Modify the ironic ports REST API to support the `physical_network` field.
- Modify the openstack baremetal CLI to support the `physical_network` field.
- Modify the ironic `VIFPortIDMixin` plugin with the new port mapping algorithm.
- Modify the ironic `NeutronNetwork` network driver to be physical network- aware when creating neutron ports for cleaning and provisioning.
- Modify the ironic network drivers to add the physical network to neutron ports binding profiles.
- Add support for multiple (virtual) physical networks to DevStack.
- Update the ironic developer documentation to cover the use of physical networks.

Dependencies

None

Testing

Support will be added to DevStack for ironic environments with multiple (virtual) physical networks.

Upgrades and Backwards Compatibility

The proposed data model and algorithm changes are backwards compatible. A database migration will be provided to add the `physical_network` field to existing ports with a null value.

Documentation Impact

The ironic developer documentation will be updated to cover the use of this feature.

References

5.15.185 Power fault recovery

https://storyboard.openstack.org/#!/story/1596107

This spec proposes adding a new node field to reflect any faults detected by the system. For nodes put into maintenance mode due to power-sync failures, this proposes a mechanisms to try to recover the nodes from the failure.

Problem description

Currently, ironics API and data model do not differentiate between ironic setting maintenance on a node (for instance, when cleaning fails, or when the power status loop cannot contact the BMC) and an operator setting maintenance on a node (an explicit API action).

This situation makes it difficult to apply any power fault recovery mechanism.

Proposed change

A string field named `fault` will be added to ironics data model for a Node. It will be used to record any detected faults.

The field will be set to one of the following values, depending on the situation (when ironic puts the node into maintenance due to one of these):

- `power failure`: when a node is put into maintenance due to power sync failures that exceed max retries.
- `clean failure`: when a node is put into maintenance due to failure of a cleaning operation.
- `rescue abort failure`: when a node is put into maintenance due to failure of cleaning up during rescue abort.
- None: there is no fault detected for the node; this is the default value. Since the field is set to a fault when ironic puts the node into maintenance due to a fault, it gets set to None when the node is taken out of maintenance (by ironic or the user).

  For nodes that were in maintenance prior to upgrading to this release, the field will also be None, since we dont know for sure, if (or what) there had been a fault. Even if there had been, we dont know whether the operator wants the node in maintenance for other reasons as well.

A periodic task will be added to the conductor. This will operate on nodes with `node.fault == power failure`. It will try to get the nodes power state. If successful, the node will be taken out of maintenance.

An integer configuration option named `[conductor]power_failure_recovery_interval` will be added. This is the interval (number of seconds) between each run of the periodic task. The default value is 300 seconds (5 minutes). Setting it to 0 will disable power fault recovery.

Alternatives

Proposals have already been made and rejected to use a combination of maintenance being set and a given maintenance_reason or last_error to do self-healing due to inconsistency.

Specific faults support[1] is proposed but its too big and to-date, no consensus has been reached on it.

An earlier version of this spec had proposed a `maintenance_type` field for the node, instead of a `fault` field. We prefer the `fault` field since it better reflects that it is about faults. `maintenance_type` caused the field to be tied in with the existing `maintenance` and `maintenance_reason` fields. Using `fault` makes it distinct from `maintenance`; it will make it easier to move forward in the future, should we decide to provide more enhancements that are fault-related, or to separate the notion and handling of faults from maintenance (which some would argue should only be operator-initiated).

[1] https://review.opendev.org/#/c/334113/

Data model impact

A string field named `fault` will be added to ironics node table.

The field will be added to `Node` object; object version will be incremented.

State Machine Impact

None

REST API impact

The field `fault` will be added to `Node` API object, ironic API version will be bumped.

Node-related API endpoints will be affected:

- POST /v1/nodes
- GET /v1/nodes
- GET /v1/nodes/detail
- GET /v1/nodes/{node_ident}
- PATCH /v1/nodes/{node_ident}

For requests with older microversion, `node.fault` is hidden from the response. This will be done in the `hide_fields_in_newer_versions()`.

Field `fault` is read-only from the API, and can only be modified internally. Any POST/PATCH request to Node with `fault` set will get 400 (Bad Request).

There is no other impact to API behaviors.

Client (CLI) impact

ironic CLI

None

openstack baremetal CLI

The CLI will be updated to support field `fault`, guarded by ironic API microversion.

RPC API impact

None

Driver API impact

None

Nova driver impact

None, although we should update the ironic-virt driver code to also look at this new node.fault field, in addition to the `node.maintenance` field.

Ramdisk impact

None

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

If the periodic task for recovery is enabled, it will consume some resources on conductor node.

More will be consumed in an environment containing some nodes in maintenance due to power failure.

Other deployer impact

A new option `[conductor]power_failure_recovery_interval` is introduced to support power failure recovery. The default value is 300 (5 minutes), you have to set to 0 if this feature is not needed.

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
kaifeng

Other contributors:
dtantsur

Work Items

- Update db layer to include the `fault` field.
- Change places where nodes enter/leave maintenance, to set `fault` accordingly.
- Add `[conductor]power_failure_recovery_interval` option to ironic configuration, add periodic task to handle power recovery.
- API change.

Dependencies

None

Testing

The feature will be covered by unit tests, API change will be covered by tempest test.

Upgrades and Backwards Compatibility

ironic API change is guarded by microversion.

When upgrading, any nodes in maintenance will have `node.fault` set to None. This is because there is no easy/guaranteed way to determine if a node had been (previously) put into maintenance due to one of our internal faults. Furthermore, even if we could determine whether it was due to a fault, we dont know whether the operator wants to keep the node in maintenance for other reasons besides the fault.

Should the operator want to take advantage of the power fault recovery mechanism, they could take the nodes out of maintenance. If there are still issues, ironic will do its thing detect the fault and try to recover.

Documentation Impact

- Update api-reference.
- New option will be generated by config generator.

References

5.15.186 Apply pre-defined system hardware configuration in single step

https://storyboard.openstack.org/#!/story/2003594

Ironics popularity and the number of ironic deployments, both integrated OpenStack and stand-alone, continue to grow. Alongside that, the number of bare metal systems managed by each deployment is increasing. Those trends have led operators to demand that improvements be made to the operational management of hardware system configuration BIOS settings, RAID, boot mode, network ports, inventory, and more. Solutions must reduce operational costs and the time required to fulfill a tenants request for a newly deployed and activated bare metal system.

Use cases which would benefit from this proposal include:

- Correct, consistent, and rapid hardware configuration during deployment of fleets of bare metal systems with similar hardware capabilities
- Production environments in which physical systems are repurposed for workloads that require different configurations and fast turnaround is at a premium
- Inventory of individual system configurations

To achieve that, prospective solutions should offer:

- Creation of bare metal system configurations from common or golden systems
- Management of configurations stored at a designated storage location
- Application of a stored configuration to a bare metal system
- Acquisition of a systems resulting entire configuration following application of a complete or partial configuration

- Acceleration of hardware configuration by reducing the amount of time, often via fewer required reboots, which is dependent on system and ironic hardware type support

This specification proposes a generally applicable solution comprised of new cleaning and deploy steps which process system configurations. Each hardware type may implement it according to its and its systems capabilities.

Problem description

A large number of systems with identical or similar hardware arrives. The user wants all of their hardware configured the same.

- In ironic, that can be a tedious, error-prone task.
- Ironic currently has APIs to configure BIOS and RAID, but it does not offer interfaces for all subsystems for which vendors offer configuration.
- Applying a configuration can take a great deal of time, often requiring multiple reboots.

The same challenges apply when different workflows are run on the same system, each needing a different hardware configuration.

Proposed change

Introduction

The following concept is used in this specification:

configuration mold
: A document which expresses the current or desired configuration of a hardware system. It also offers an inventory of a systems hardware configuration.

This specification proposes to:

- Add new, optional cleaning[1] and deploy[2] steps which, in a single step, can completely configure a systems hardware. They can apply the saved configuration of a system with sufficiently similar hardware capabilities. Steps which obtain and persistently store a systems configuration are also proposed.
  - `export_configuration`
    : obtain a systems hardware configuration and persistently store it as a *configuration mold* so it can be used to configure other, sufficiently capable systems and as an inventory. More details are available in *Export configuration*.
  - `import_configuration`
    : apply the hardware configuration described by an existing, persistently stored *configuration mold* to a sufficiently capable system. More details are available in *Import configuration*.
  - `import_export_configuration`
    : import and then export the hardware configuration of the same system as a single, atomic step from the perspective of ironics cleaning and deployment mechanisms. This can further reduce the time required to provision a system than a sequence of separate import and export steps. More details are available in *Import and export configuration*.

[1] https://docs.openstack.org/ironic/latest/admin/cleaning.html#cleaning-steps
[2] https://docs.openstack.org/ironic/latest/admin/node-deployment.html#node-deployment-deploy-steps

- Define the data format of the vendor-independent content of a *configuration mold*. It contains a section for each hardware subsystem it supports, RAID and BIOS. The desired configuration of a subsystem for which ironic has been offering configuration support, again RAID and BIOS, is expressed as much as possible in ironic terms.

  To support hardware configuration beyond what ironic defines, but which vendors, systems, and their hardware types could offer in a system-specific manner, a third section, OEM, is proposed. The OEM section can also be used to specify configuration ironic already supports. The content of the OEM section is specific to the system and its hardware type.

  More details, along with an example, are in section *Format of configuration data*.

- Describe the persistent storage of *configuration molds* for both integrated OpenStack and stand-alone ironic environments and how ironic and its users interact with them. See *Storage of configuration data*.

- Declare what is considered a complete implementation for the purposes of this specification. An implementation will be considered complete if it offers all three (3) pairs of cleaning and deploy steps and supports all of the defined *configuration mold* sections, presently BIOS and RAID. Additionally, persistent storage of *configuration molds* must be supported for both ironic environments, integrated OpenStack and stand-alone.

- Offer an alternative, not a replacement, to ironics currently available granular configuration steps which operate on a single subsystem: RAID and BIOS. How it compares to ironics present functionality is described in *Alternatives*.

- Outline a minimum viable product (MVP) implementation by the `idrac` hardware type in section *idrac-redfish implementation*. We aim for it to offer all three (3) pairs of cleaning and deploy steps. In the MVP, only the OEM *configuration mold* section will be supported. Both methods of persistent *configuration mold* storage will be implemented and supported.

- Illustrate possible implementation for generic `redfish` hardware type in section *redfish implementation*.

Goals

With this specification, we are going to achieve the following goals:

- Define an ironic hardware type framework which offers more operationally efficient bare metal hardware configuration

- Facilitate a consistent method to accelerate bare metal hardware configuration, necessitating fewer reboots, when supported by a hardware system and its ironic hardware type

- Describe a first, MVP implementation by the `idrac` hardware type

Non-goals

The following are considered outside the scope of this specification:

- Implementation of the approach by all hardware types; it is optional

- Requiring a hardware types implementation be complete; it may be partial

Export configuration

The export configuration clean/deploy step extracts existing configuration of indicated server (golden server) and stores it in designated storage location to be used in *Import configuration* clean/deploy step.

Clean/deploy step details are:

Interface
Management interface

Name
`export_configuration`

Details
Gets the configuration of the server against which the step is run and stores it in specific format in indicated storage as configured by ironic.

Priority
0

Stoppable
No

Arguments

- URL of location to save the configuration to

Sample of clean/deploy step configuration:

```
{
  "interface": "management",
  "step": "export_configuration",
  "args": {
    "export_configuration_location": "https://server/edge_dell_emc-poweredge_
↪r640"
  }
}
```

The workflow of configuration export consists of 3 parts:

1. Get current nodes configuration (driver specific)
2. Transform the configuration to common format (transformation is driver specific; format is common, see *Format of configuration data*)
3. Save the storage item to designated storage (common to all drivers, see *Storage of configuration data*)

Usage of *export_configuration* is not mandatory. If the configuration is acquired previously or in another way, user can also upload it to storage directly.

Import configuration

Once the configuration is available, user can use it in the import configuration clean/deploy step to configure the servers.

Clean/deploy step details are:

Interface
Management interface

Name
`import_configuration`

Details
Gets pre-created configuration from storage by given location URL and imports that into given server.

Priority
0

Stoppable
No

Arguments

- URL of location to fetch desired configuration from

Sample:

```
{
  "interface": "management",
  "step": "import_configuration",
  "args": {
    "import_configuration_location": "https://server/edge_dell_emc-poweredge_
↪r640"
  }
}
```

The workflow of the import configuration consists of 3 parts:

1. Using given configuration location and ironics storage settings, get the configuration from the storage (common to all drivers).
2. Transform the configuration to driver specific format (driver specific)
3. Apply the configuration (driver specific)

- Sections that are not specified in the *configuration mold* are left intact, for example, it is possible to configure only subset of BIOS settings and other BIOS settings and RAID settings remain unchanged.
- If an error is encountered, the clean/deploy step fails. On failure, no assurances can be made about the state of the systems configuration, because the application of the configuration is system and ironic hardware type dependent and there are many possible failure modes. A defined subsystem configuration sequence and transactional rollback semantics do not seem to apply.
- When a step fails, the ironic node is placed in the `clean failed` or `deploy failed` state and the nodes `last_error` field may contain further information about the cause of the failure.
- Successful application of configuration specified has no side effects on the nodes fields (like BIOS and RAID configuration).

Warning

Depending on each vendors capabilities importing can be powerful step that allows configuring various things. Users and vendors need to be aware of these capabilities and make sure not to overwrite settings that are not intended to be replaced, for example, deleting RAID settings or static BMC IP address.

Import and export configuration

Import and export configuration clean/deploy step is composite step that executes both importing and exporting one after another as atomic operation. This can be used to get the inventory just after configuration and can be useful when not all aspects of system are being configured, but need to know the outcome for all aspects.

Clean/deploy step details are:

Interface
Management interface

Name
`import_export_configuration`

Details
Gets pre-created configuration from storage, imports that into given server and exports resulting configuration.

Priority
0

Stoppable
No

Arguments

- URL of location to fetch desired configuration from
- URL of location to save the configuration to

Sample of clean/deploy step configuration:

```
{
  "interface": "management",
  "step": "import_export_configuration",
  "args": {
      "import_configuration_location": "https://server/edge_dell_emc-
↪poweredge_r640"
      "export_configuration_location": "https://server/edge_dell_emc-
↪poweredge_r640_server005"
  }
}
```

The workflow of configuration import and export consists of parts:

1. Execute workflow as in step *Import configuration*
2. When importing succeeds, execute workflow as in step *Export configuration*

Format of configuration data

The format to store the reusable configuration is in JSON format and consists of 3 sections:

- `bios reset` to indicate if reset is necessary before applying settings indicated in the list of BIOS attribute key-value pairs inside `settings` section as in Apply BIOS configuration step[3]. If `reset` is false, then settings that are not included in `settings` sections are left unchanged.
- `raid` as in RAID create configuration step with key-value pair settings and `target_raid_config` property[4]
- `oem` driver specific section with everything else that does not fit into bios and raid sections together with interface name that can handle this data. The interface name can be used to distinguish for which hardware type this configuration data is meant and used for validation during import before trying to parse this section and catch incompatibility early. The data format of this section is controlled by implementing interface and only restriction is that it needs to fit in JSON property.
- There is no overlapping with `oem` and vendor-independent sections, like `bios` and `raid`.
- If overlapping is determined during import, then configuration data is considered invalid and cleaning/deployment step fails.

Sample of exported data format:

```
{
  "bios": {
    "reset": false,
    "settings": [
      {
        "name": "ProcVirtualization",
        "value": "Enabled"
      },
      {
        "name": "MemTest",
        "value": "Disabled"
      }
    ]
  }
  "raid": {
    "create_nonroot_volumes": true,
    "create_root_volume": true,
    "delete_existing": false,
    "target_raid_config": {
      "logical_disks": [
        {
          "size_gb": 50,
          "raid_level": "1+0",
          "controller": "RAID.Integrated.1-1",
          "volume_name": "root_volume",
          "is_root_volume": true,
          "physical_disks": [
            "Disk.Bay.0:Encl.Int.0-1:RAID.Integrated.1-1",
```

(continues on next page)

[3] https://docs.openstack.org/ironic/latest/admin/bios.html#apply-bios-configuration
[4] https://opendev.org/openstack/ironic/src/branch/master/ironic/drivers/raid_config_schema.json

(continued from previous page)

```
            "Disk.Bay.1:Encl.Int.0-1:RAID.Integrated.1-1"
          ]
        },
        {
          "size_gb": 100,
          "raid_level": "5",
          "controller": "RAID.Integrated.1-1",
          "volume_name": "data_volume",
          "physical_disks": [
            "Disk.Bay.2:Encl.Int.0-1:RAID.Integrated.1-1",
            "Disk.Bay.3:Encl.Int.0-1:RAID.Integrated.1-1",
            "Disk.Bay.4:Encl.Int.0-1:RAID.Integrated.1-1"
          ]
        }
      ]
    }
  }
  "oem": {
    "interface": "idrac-redfish",
    "data": {
      "SystemConfiguration": {
        "Model": "PowerEdge R640",
        "ServiceTag": "8CY9Z99",
        "TimeStamp": "Fri Jun 26 08:43:15 2020",
        "Components": [
          {
            [...]
            "FQDD": "NIC.Slot.1-1-1",
            "Attributes": [
              {
              "Name": "BlnkLeds",
              "Value": "15",
              "Set On Import": "True",
              "Comment": "Read and Write"
              },
              {
              "Name": "VirtMacAddr",
              "Value": "00:00:00:00:00:00",
              "Set On Import": "False",
              "Comment": "Read and Write"
              },
              {
              "Name": "VirtualizationMode",
              "Value": "NONE",
              "Set On Import": "True",
              "Comment": "Read and Write"
              },
            [...]
            ]
```

(continues on next page)

(continued from previous page)

```
            }
          ]
        }
    }
}
```

`oem` section of sample data depicts snippets from Dell SCP file (see more at *idrac-redfish implementation*) that has some metadata about the source of the configuration (`Model`, `ServiceTag`, `TimeStamp`) and inside `Components` section there are attributes listed that can be applied during import and is controlled by `Set On Import` property.

Storage of configuration data

Common functionality among hardware types is the configuration storage and will be implemented for all vendors to be used in their implementations.

The requirements for storage are:

1. Support node multi-tenancy
2. Support multiple conductors
3. Support ironic in stand-alone mode
4. Support ironic operators with ephemeral ironic database
5. User can manage (list, create, view, edit, delete) *configuration molds*

To fulfill these requirements a storage solution is proposed:

- Use full URL to indicate *configuration mold* location
- Swift is used as storage backend
  - Full URL points to Swift object within containers
  - Access is restricted by projects/tenants/accounts
  - HTTP GET and HTTP PUT used to get and store data
  - Ironic service account has access to used containers
  - User can manage *configuration molds* by accessing Swift container directly
- Web server is used as storage backend for ironic in stand-alone mode
  - Used only for Ironic stand-alone or when there is only one tenant as this does not guard against accessing other tenant data
  - HTTP GET and HTTP PUT used to get and store data. Web servers need to have HTTP PUT configured (it is not enabled by default)
  - Basic auth used with pre-defined credentials from ironic configuration to access data
  - User can manage *configuration molds* by accessing the web server directly
- In future additional storage back-ends can be added

To implement this, these changes will be made:

New settings added:

```
[molds]storage
[molds]user
[molds]password
```

`[molds]storage` used to define what storage backend used. By default it will be Swift with option to configure Web server. In future more options can be added.

`[molds]user` and `[molds]password` is used when Web server is configured as storage backend. Ironic will use this to encode it in Base64 and add as header to HTTP requests. By default they will be empty indicating that no authorization used.

The workflow for getting stored configuration data:

1. Given *configuration mold*s full URL and used storage mechanism configured in `[molds]storage` fetch data using appropriate credentials.
2. Handle any errors, including access permission errors. If errors encountered, a step fails.

The workflow for storing the configuration data:

1. Given *configuration mold*s full URL and used storage mechanism configured in `[molds]storage` store data using appropriate credentials.
2. Handle any errors, including access permission errors. If errors encountered, a step fails.

Swift support

For Swift as end-user that initiates cleaning or deploying is different from the service user that actually does cleaning or deploying, it is necessary to allow ironic conductor (service user) access Swift containers used in steps. There is security risk having access to all tenant containers that is described in *Security impact* section.

Web server support

For web server authorization Basic authentication will be used from `[molds]user` and `[molds]password`. It is strongly advised to have TLS configured on the web server.

idrac-redfish implementation

For iDRAC to implement these proposed steps it will use Server Configuration Profile (SCP)[5] that allows to export existing configuration server and import the same configuration file to another server. Settings for different sub-systems such as BIOS, RAID, NIC are included in the configuration file.

The implementation would transform configuration between SCP data format and ironic data format. In the first version (MVP), all SCP data is exported to and imported from `oem` section as-is without any transformation. In the following versions this will be improved to start using `bios` and `raid` sections. The implementation will use Redfish protocol. As this is part of OEM section in Redfish service, the communication will be implemented in sushy-oem-idrac library. In next versions after MVP is done, transformation between SCP data format and ironic data format will be implemented in ironic part of idrac-redfish interface.

[5] https://downloads.dell.com/manuals/all-products/esuprt_solutions_int/esuprt_solutions_int_solutions_resources/dell-management-solution-resources_white-papers15_en-us.pdf

When comparing configuration runtime using separate BIOS and RAID configuration jobs versus SCP approach on R640 the difference was 11 minutes versus 7 minutes where SCP was faster within one reboot.

redfish implementation

This section describes how these steps could be implemented for generic `redfish` driver. Compared to the proposed implementation of `idrac-redfish` described in *idrac-redfish implementation* that implements these steps using iDRAC specific functionality, Redfish specification, in contrast, does not have a resource and action that accepts configuration data all together and implementation for `redfish` needs to take different approach.

This illustrates how dedicated resources as they are available in Redfish service can be assembled to be used in single step. It is not part of this spec to implement this.

The following describes how this implementation would support configuration of RAID, BIOS.

For `import_configuration`:

1. Given *configuration mold* that contains `bios` and `raid` sections.
2. Apply RAID configuration to create a volume with immediate application or apply on reset (if changes need reboot).
3. Apply BIOS updates to pending settings with apply time on reset.
4. Reboot the system for pending changes to take effect.

The difference from existing functionality in ironic:

1. There is only 1 step instead of dedicated 2 steps.
2. There are less reboots (depending on necessity to reboot for RAID) as all configuration is assembled before reboot.

For `export_configuration`:

1. Use BIOS resource to get current BIOS settings.
2. Use Storage resource and related to get current RAID settings.
3. Transform results into *configuration mold*s format.

The difference from existing functionality in ironic:

1. There is no step that fetches current configuration across various subsystems.
2. Closest to achieve this would be getting nodes `raid_config` field and getting BIOS attributes and transforming it to deploy template that uses existing RAID and BIOS steps.

For `import_export_configuration` combine implementations of `import_configuration` and `export_configuration` together.

Alternatives

We can continue to support only the current, granular hardware provisioning deploy and clean steps.

The closest currently available functionality in ironic is deploy templates that enable assembling several existing steps together. In the same manner these deploy templates can be re-used for as many systems as necessary. However, comparing deploy templates to the proposed solution currently:

- No functionality to get the configuration from already configured system, user has to construct the initial configuration file themselves by hand or a script. To make it easier, can use cached BIOS and RAID settings from a node that was deployed, but this reuse is still not built in ironic.
- Depending on vendors capabilities each step may require reboot to finish. For example, iDRAC BIOS configuration apply needs reboot to take effect and deem the step to be finished. For now ironic cannot line up several steps that require reboot and then finish them all by one reboot. For next step to start the previous one needs to be finished. The proposal makes it possible to handle this internally inside the import step, that is, if that is how a hardware type is implementing this, it can create 2 tasks for BIOS, RAID configuration and then reboot and watch for both tasks to finish to deem the step as finished.
- Using OEM section each vendor can add support for configuring more settings that are not currently possible using common (vendor-independent) ironic clean/deploy steps.

This proposal does not suggest to replace current clean/deploy steps and deploy templates but add alternative approach for system configuration.

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

None

Security impact

JSON will be used as user input. It will be validated, sanitized, and treated as text. Common storage utils will use Pythons *json.loads* when retrieving and *json.dumps* when storing data. If there is additional validation and clean up necessary for vendor specific implementation, for example, OEM section, then that needs to be added to drivers implementation.

Configuration molds are considered a sensitive data and they also can contain plain text password, for example, for BMC. Implementation for storage of *configuration molds* will support authorization and separation of tenants. Operators will be suggested to add additional security e.g., configure storage back-end encryption and TLS for web server.

As cleaning/deploying is executed by ironic service account and not user that initiated clean or deploy, ironic service account needs access to used Swift containers provided by users. In multi-tenant, end-user accessible Ironic API this could lead to accessing data not belonging to the tenant by, e.g., guessing or somehow finding out another tenants URL and feeding that to ironic that has access to it while end-user does not. This issue needs to be addressed separately in future releases. There are possibly other use cases that are affected by this limitation and would benefit from addressing this.

Other end user impact

The configuration items can accumulate in the storage as there is no default timeout or logic that deletes them after a while because these configuration items should be available after nodes cleaning or deployment. If user do not need the reusable configuration items anymore, then user should delete those themselves from the storage.

This adds new configuration section `[molds]` to control storage location. Default values are provided.

Scalability impact

None

Performance Impact

Depending on hardware type implementation, deployments can become faster. When *configuration mold* is processed, it is read in memory, but it is not expected that these *configuration molds* will be large.

Also based on vendors implementation these can be synchronous or asynchronous steps. If steps are synchronous this will consume a long-lived thread where operators may need to adjust the number of workers.

Other deployer impact

Need to configure storage backend, Swift or web server.

Developer impact

There will be new clean and deploy steps available that each driver can implement. They are optional and other developers can implement those at their own time if needed.

Implementation

Assignee(s)

Primary assignees:

- Aija Jaunteva (@ajya, aija.jaunteva@dell.com)
- Richard Pioso (@rpioso, richard.pioso@dell.com)

Other contributors:
None

Work Items

For common functionality:

- Implement common functionality for configuration storage
  - Swift support
  - Web server support

For `idrac-redfish` implementation:

- Implement initial idrac hardware type derivations of the new clean and deployment steps which use the Redfish protocol (MVP)
- Update the iDRAC driver documentation
- Enhance the idrac hardware type implementation to support the `bios` section of the configuration data
- Enhance the idrac hardware type implementation to support the `raid` section of the configuration data

Dependencies

None

Testing

For now, tempest tests are out of scope, but in future 3rd party continuous integration (CI) tests can be added for each driver which implements the new clean and deploy steps.

Upgrades and Backwards Compatibility

This change is designed to be backwards compatible. The new clean and deploy steps are optional. When an attempt to use them with a hardware type which does not implement them, then clean or deploy will fail with error saying that node does not support these steps.

Documentation Impact

Node cleaning documentation[6] is updated to describe new clean steps under Management interface for `idrac-redfish`.

[6] https://docs.openstack.org/ironic/latest/admin/cleaning.html

References

5.15.187 Ceph Object Gateway Temp URL support

https://bugs.launchpad.net/ironic/+bug/1526395

This adds support of Ceph Object Gateway (RADOS Gateway) temporary URL format.

Problem description

Ceph project is a powerful distributed storage system. It contains object store with OpenStack Swift compatible API. Glance image service can use Ceph storage via RADOS Gateway Swift API. Ironic does not currently support deploy configuration with Glance and RADOS Gateway. The reason is different format of temporary URL. First part of temporary URL for RADOS Gateway with Glance frontend is endpoint_url/api_version/container/object_id, where

- endpoint_url contains scheme, hostname, optional port and mandatory /swift suffix.
- api_version is v1 currently.
- container is the name of Glance container.
- object_id is Glance object id.

Calculation of parameters temp_url_sig and temp_url_expires is mostly the same as in Swift, so full URL looks like

https://radosgw.my.host/swift/v1/glance/22aee8e5-cba3-4554-92c4aadde5e38f28?
temp_url_sig=e75d1d6facb53d795547b1fc60eca4e8836bd503 &temp_url_expires=1443518434

temp_url_sig calculation should not use /swift in the path.

OpenStack Swift temporary URL contains extra account parameter, its account that Glance uses to communicate with Swift. swift_account parameter is mandatory for Ironic.

> **Note**
>
> Do not use Python code as reference from http://docs.ceph.com/docs/master/radosgw/swift/tempurl/ it does not create valid URLs, for Firefly release at least.

Proposed change

A new configuration parameter `temp_url_endpoint_type` will be added to the `glance` group. It can be set to values swift or radosgw, swift is default. Code of image service in Ironic will be changed for supporting both of endpoints (parameters set, mandatory suffix for RADOS Gateway).

Alternatives

None

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

N/A

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

- A new config option `temp_url_endpoint_type` will be added in `glance` group.
- Deployer should configure Glance with RADOS Gateway beckend (via Swift API) and Ceph storage.

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
yuriyz

Work Items

- Implement Rados GW support.
- Add unit tests.

Dependencies

None

Testing

Unittests will be added.

Upgrades and Backwards Compatibility

None

Documentation Impact

Usage of Ironic with Rados Gateway as Glance backend will be documented.

References

- http://docs.openstack.org/kilo/config-reference/content/object-storage-tempurl.html

5.15.188 Allow a ramdisk deploy user to specify their boot ISO

https://storyboard.openstack.org/#!/story/2007633

With support for virtual media, there are cases where an operator may wish to boot a machine with a specific virtual media image to facilitate the deployment of a machine or even just the completion of an action like firmware upgrades.

Providing an interface to signal boot to this iso, seems logical.

Problem description

A detailed description of the problem:

- The `ramdisk` deployment interface allows a user to define a kernel and ramdisk to be utilized to boot a ramdisk instance which after the initial deployment the instance is considered a deployed machine in active state.
- At present, because of the `ramdisk` deployment interface constraints, users are unable to specify ISOs for virtual media. They must supply a kernel/ramdisk and in the virtual media case it must be rebuilt.

Proposed change

- Allow a `instance_info/boot_iso` parameter to be leveraged to be the medium utilized for booting from an explicit ISO image which the conductor would support downloading, caching, and providing to the baremetal machine. This change would be focused on the virtual media boot interface code in relation to usage of Redfish.
- Teach the code related to the pass-through to provide the same basic capability to append parameters to the command line through decomposition of the ISO, appending to `grub2` and `isolinux` configurations with the supplied values, and repackaging of the ISO file for deployment.

Alternatives

- Use an external provisioning tool, and `adopt` the node into Ironic, if applicable.
- Pre-mastered machine specific configurations into ISO images which would ultimately result in pushing the preparation and execution workload to the API user.

Data model impact

None, this leverages the existing data model.

State Machine Impact

None

REST API impact

None, this change leverages existing JSON data fields with-in Ironics data model.

Client (CLI) impact

openstack baremetal CLI

None

openstacksdk

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

None

Security impact

None

Other end user impact

None

Scalability impact

The distinct possibility exists, if a user requests multiple concurrent deployments, that configuration injection could consume a large amount of disk space.

Also, we may wish to enable some sort of logic to prevent mass consumption of disk space as part of the conductor, for the reasons of cleanup, however the conductor has no real way to understand if this is a forever usage, or not. Ideally operator documentation would be updated to help scale planning for this condition. Alternatively we may wish to introduce a one-shot indicator flag so we dont attempt to persist ISOs after takeover on active machines.

Performance Impact

A large number of concurrent deployments may slow the conductor due to overall system performance constraints, depending on the exact options and settings leveraged.

Other deployer impact

None

Developer impact

None is anticipated, however we would likely focus on implementing this in the redfish virtual media code path, and should likely try to ensure that we do not make such changes redfish interface specific as other drivers are present in Ironic which support Virtual Media.

Implementation

Assignee(s)

Primary assignee:
Julia Kreger <juliaashleykreger@gmail.com>

Work Items

- Implement support to pass an explicit boot ISO into the ramdisk interface.
- Implement support to inject configuration into the boot ISO.
- Document this functionality for the ramdisk interface covering how to leverage this feature.

Dependencies

- None

Testing

Unit tests should be sufficient for ensuring this functionality is not broken.

A tempest test may also be viable, but we may wish to partner with the Metal3 community on integration testing, as ultimately this is essentially just an item of integration testing when virtual media AND ramdisk interfaces are leveraged.

Upgrades and Backwards Compatibility

N/A

Documentation Impact

We will want to update the documentation on the ramdisk deployment interface to detail this capability.

References

None

5.15.189 Out of Band Inspection support for `redfish` hardware type

https://bugs.launchpad.net/ironic/+bug/1668487

This proposal adds the ability to inspect/update hardware properties and auto-create ports in OOB manner for Redfish hardware type.

Problem description

Node inspection automatically collects and updates node properties. These properties can be used to seggregate bare metal nodes into appropriate resource classes that could be used in nova scheduling. Node inspection also creates ironic ports for all discovered NIC(s). The `redfish` hardware type has support for inband inspection using `inspector` (Ironic Inspector). The DMTF standard Redfish schemas supports OOB interfaces to fetch most of the inspection properties supported by ironic. By adding support for OOB inspection, the overall time needed to introspect a bare metal can be reduced.

Proposed change

This spec proposes add OOB inspection support for `redfish` hardware type. It would discover ironic supported node properties and capabilities for Redfish compliant servers. This will be done by enhancing Sushy library to fetch the required properties from Redfish controller running on the bare metal BMC.

The following mandatory properties will be discovered and updated in `node.properties` for `redfish` hardware type, as discussed in Introspect spec

- memory size
- CPUs
- CPU architecture
- NIC(s) MAC address
- disks

It would also implement the additional capabilities discussed in Common Ironic Capabilities spec and available using DMTF standard Redfish schemas for `redfish` hardware type.

The properties which are already set will be overridden upon invocation of `inspect_hardware()` except for ironic ports. If a ironic port already exists, it will not create a new port for that MAC address. It will take care of adding as well as deleting of the ports for NIC changes as discussed in Introspect spec. Not all the capabilities supported by ironic are available in all Redfish compliant servers. If a property is not available in the hardware, the property will not be added/updated in node.properties as capabilities.

Inspection would return failure in the following cases:

- Failed to get basic properties.
- Failed to get capabilities, due to service configuration errors.
- Communication errors with Redfish manager.

Sushy changes

Implement `InspectInterface` method `inspect_hardware` in Sushy library.

Alternatives

One can continue to discover these properties using inband mechanism of `inspector` supported by `redfish` hardware type.

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

ironic CLI

None

openstack baremetal CLI

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

None

Security impact

None

Other end user impact

With OOB inspection, time required for hardware introspection would be reduced.

Scalability impact

None

Performance Impact

None

Other deployer impact

With OOB inspection, time required for hardware introspection would be reduced.

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
stendulker

Other contributors:
agarwalnisha1980

Work Items

- Implementation of the `InspectInterface` class and its methods `inspect_hardware()`, `validate()` and `get_properties()`.
- Enhance Sushy library to discover required hardware properties.

Dependencies

- Depends on Sushy library

Testing

- Unit tests will be added conforming to ironic testing requirements.
- CI support will be added for inspection server using virtual CI based on sushy-tools.

Upgrades and Backwards Compatibility

None

Documentation Impact

Redfish hardware type document would be updated for OOB inspection feature.

References

- Introspect spec
- Common Ironic Capabilities spec
- Sushy library
- Redfish hardware type

5.15.190 Self-Service via Runbooks

https://bugs.launchpad.net/ironic/+bug/2027690

With the addition of service steps, combined with owner/lessee, we now have an opportunity to allow project members to self-serve many maintenance items by permitting them access to curated runbooks of steps.

This feature will primarily involve extending creating a new runbook concept, allowing lists of steps to be created, associated with a node via traits. These runbooks will then be able to be used in lieu of a list of steps when performing manual cleaning or node servicing.

Problem description

Currently, users of the Ironic API as a project-scoped member have limited ability to self-serve maintenance items. Ironic operators are given the difficult choice of giving users broad access to nodes, allowing them to run arbitrary manual cleaning or service steps with the only alternative being permitting no access to self-serve these maintenance items.

Use cases include:

- As a project member, I can execute runbooks via Node Servicing without granting the ability to execute arbitrary steps on a node.

- As an system manager, I want to store a list of steps to perform an action in an identical manner across many similar nodes.

Proposed change

The proposed change is to create a new API concept, runbooks, which can be used with any API flow which currently takes explicit lists of steps.

Those runbooks can then be used instead of a list of clean_steps or service_steps[0] when setting node provision state. These are expected to behave identical to API calls with `clean_steps` or `service_steps` provided, including honoring the `disable_ramdisk` field, and providing explicit ordering rather than the priority-based ordering that is used in automated cleaning and deploys.

Additionally, we will ensure that the full CRUD lifecycle of runbooks is made role-aware in the code, so that a project can limit who can create, delete, edit, or mark runbooks as public all as separate policy toggles. We will also ensure deployers can separately toggle the ability to run step-based flows via runbooks versus step-based flows with arbitrary step lists.

A runbook will only run on a node who has a trait equal to the runbook name, to ensure the runbook has been approved for use on a given piece of hardware, as an extra precaution against hardware breakage.

Alternatives

We considered, originally, repurposing the existing deploy templates into a generic concept of templates. This was abandoned due to deploy templates containing implicit steps, making it difficult to reason about them. This is why we instead chose to call them runbooks, which are entirely specified as opposed to templates, which are partially specified and have implicit steps integrated.

Data model impact

Create new tables described below:

```
``runbooks`` (same as ``deploy_templates`` except addition of ``owner`` and␣
↪``public``)
  - id (int, pkey)
  - uuid
  - name (string 255)
  - public (bool) - When true, template is available for use by any project.
  - owner (nullable string, usually a keystone project ID)
  - disable_ramdisk - When true, similar behavior to disable_ramdisk in␣
↪manual cleaning -- do not boot IPA
  - extra json/string
  - steps list of ids pointing to ``runbook_steps``

``runbook_steps``
  - id (int, pkey)
  - runbook_id (Foreign Key to runbooks.id)
  - interface
  - step
  - args
  - order (or some other field/method to indicate how the steps were ordered␣
↪coming into the API)
```

[0] *Change Node Provision State*: https://docs.openstack.org/api-ref/baremetal/#change-node-provision-state

Note: Ensure all queries to `runbooks` only pull in `runbook_steps` if needed.

State Machine Impact

While no states or state transitions are being proposed, the APIs to invoke some of those state transitions will need to change to become runbook-aware.

REST API impact

A new top level REST API endpoint, `/v1/runbooks/` will be added, with basic CRUD support.

The existing `/v1/nodes/<node>/states/provision` API will be changed to accept a `runbook` (name or uuid) in lieu of `clean_steps` when being used for servicing or manual cleaning.

Client (CLI) impact

The CLI will be updated to add support for the new API endpoints.

Some examples of CLI commands that will be added, and how they interact with RBAC:

```
- baremetal runbook create X [opts] # as system-scoped manager
  - owner: null
  - public: false
- baremetal runbook create X [opts] # as project-scoped manager
  - owner: projectX
  - public: false
- baremetal runbook set X --public # as system-scoped manager
  - owner: null
  - public: true
  - Note: Owner field is nulled even if it previously set.
- baremetal runbook set X --public # as project-scoped manager
  - Forbidden! Requires system-scoped access.
- baremetal runbook unset X --public # as system-scoped manager
  - owner: null
  - public: false
- baremetal runbook set X --owner projectX # as system-scoped manager
  - owner: projectX
  - public: false
  - Note: Will return an error if ``runbook.public`` is true.
- baremetal node service N --runbook X
- baremetal node clean N --runbook X
- baremetal node service N --runbook X --service-steps {} # NOT PERMITTED
- baremetal node clean N --runbook X --clean-steps {} # NOT PERMITTED
```

RPC API impact

RPC API will be modified to support runbooks in lieu of steps where necessary. They will be properly versioned to ensure a smooth upgrade.

Driver API impact

None

Nova driver impact

None

Ramdisk impact

None

Security impact

Operators are warned that even with use of this feature, users may be able to leverage steps or access which are innocuous on their own, but malicious when combined.

Deployers should ensure they have reviewed all possible threat models when granting additional access to less-trusted individuals including restricting unsafe node actions, such as replacing `deploy_ramdisk` to ensure runbooks (and other step-based workflows) operate as expected.

Things for the implementer to avoid to ensure secure implementation:

- Do not permit a project-scoped API user to change `runbooks.public` by default.
- Do not permit a project-scoped API user change `runbooks.owner` by default.
- Anything that would *implicitly* mark a runbook as non-public.
- Ensure we check if nodes are able to run a given runbook using node traits, in a similar method to how we do so with deploy templates.

RBAC Impact

There are two primary ways this feature interacts with RBAC, beyond the obvious CRUD for runbooks.

First, the `runbooks.owner` and `runbooks.public` fields are relevant for determining if a runbook is scoped to a project or to a system. If `owner` is non-null and `public` is false, the runbook is scoped to the project set in that field and is only usable on nodes owned or leased by that project. If `owner` is null and `public` is false, the runbook is only able to be used or access by system-scoped users. If `owner` is null and `public` is true, a system-scoped member can modify the runbook and a project-scoped member could use it on a compatible node. Additionally, the `owner` field will only be settable when `public` is false or being set to false, and setting `public` to true will null the owner field.

Second, the node change provision state[Page 1392, 0] API will have a `runbook` field added, and policy will be different for cases where `runbook` is specified instead of `clean_steps`. Default policy will be to permit manual cleaning and servicing for a node owner or lessee-scoped member when using a runbook, but to disallow it when specifying `clean_steps`. Combining `clean_steps` and `runbook` will not be permitted.

Expected access after this implementation is complete:

```
System
- Admin
- Manager
- Member
```

(continues on next page)

(continued from previous page)

```
--> Can CRUD system-scoped templates (template.owner=null)
--> Can CRUD project-scoped templates (template.owner=PROJECT)
--> Can unset template.owner, changing a template to system-scope
--> Can mark system-scoped templates as public (template.public=True)
- Reader
--> Can list all templates

Project
- Admin
- Manager
--> Can CRUD project-scoped templates (template.owner=PROJECT)
--> Cannot set a template to public (template.public=True).
- Member
--> Can execute public templates or templates owned by their project.
- Reader
--> Can list public templates and templates owned by their project.
```

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

None

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
JayF <jay@jvf.cc>

Other contributors:
TheJulia <juliaashleykreger@gmail.com>

Work Items

- Create Runbooks - Object layer - DB layer - API layer
- Add policy checking tests for /v1/runbooks
- Ensure tempest API tests exist for new API endpoints
- Update API-Ref
- Update Manual Cleaning and Node Servicing documentation

Dependencies

All dependencies have been resolved.

Testing

Unit tests will be added to test the new functionality. Integration tests will be added to test the new API endpoints and CLI commands.

Upgrades and Backwards Compatibility

The changes are backwards compatible. Existing API endpoints will continue to function as before, and we will gate all API changes behind microversion checks.

Documentation Impact

The new functionality will need to be documented. This includes documentation for the new API endpoints and CLI commands, as well as documenting security caveats detailed above.

References

5.15.191 System Scoped Role Based Access Control

Specification Scope: OpenStack Integrated

https://storyboard.openstack.org/#!/story/2008425

Ironic has long been considered an admin-only service. This changed with recent work contributed to help make Ironic able to provide multi-tenant capabilities. This work was in support of the the efforts of the Mass Open Cloud community to support delineation of access and resource allocation between the participating organizations through an `owner` and `lessee` field.

However there is a growing desire to delineate scopes in which user accounts have access to the API. This effort is sometimes referred to as Secure RBAC in the OpenStack community, which is an initiative to have scope restricted authentication across OpenStack services, where the scoping and modeling is consistent to provide a consistent authorization experience. This is achieved via system scoped role assignments. In this model, an `admin`, `member`, and `reader` exists. The `admin` role implies `member`, and `member` implies `reader`. These roles exist in one of three scopes, `system`, `domain`, and `project`. The relevant scopes for most services in OpenStack are the `system` and `project` scopes.

In essence this effort is to group the access and actions behind personas, which are role and scope permutations that can be applied to a user via role assignments in keystone.users and then ensuring that the invoked access rights do not permit inappropriate access such as edit fields as a reader only role on the system scope. At a high level, this is conceptually modeled into `admin`, `member`, and `reader` roles. During the policy in code, effort the `admin` role was modeled the `baremetal_admin` role and the `reader`

role via the `baremetal_observer` role, however none of this is system scoped. The existing access roles are project scoped to a `baremetal` project, and Ironic has no concept of token scopes.

Role definitions:

- **admin - This is in essence an administrative user with-in the operating**
  scope. These are accounts which Create/Delete $things, and in keystone default configuration, this role implies the `member` role. In an Ironic context, we can think of this user as the infrastructure administrator who is adding their baremetal machines into Ironic.
- **member - This is a user which can act upon things. They may be able to read**
  and write with-in objects, but cannot create/delete new objects unless it is an explicitly permitted action. An Ironic example may be that we might want to permit members to be able to request allocations, or change a nodes provision state. Similar to `admin` implying `member`, `member` implies `reader`.
- **reader - This is a user which needs to be able to have read-only access.**
  They can read objects but not change, modify, or delete objects. In a `system` scope it may be a network operations center employee who has a business need to be able to observe the status and details. In a `project` scope, this may be someone attempting to account for resources, or accounts for automated processes/reporting.

Note

Additional details on default role definitions is covered in the Keystone specification define default roles or the Keystone administrators guide.

Note

A future potential is that an `auditor` role may exist, but it would *not* match readers. Auditors would be read-only in nature, but their role would likely allow sensitive values to be unmasked. This has not been decided upon, and depending on service configuration could likely be implemented manually with a custom policy file. That being said, this is out of scope of this specification document at this time. It is also important to note that the `admin`, `member`, and `reader` roles do not automatically unmask sensitive data, and should not be anticipated to do so by default.

When considering the above role definitions and the use case of baremetal, scope differences will have a major difference between what exists today and what would exist moving forward. At a high level, you wouldnt allow a `project` scoped `admin` to have full administrative access to Ironic.

Scope definitions:

- system - This is similar to the existing scope of the cloud deployment today.
- **domain - This scope, is presently only used in Keystone for associations.**
  We do not anticipate this to apply, and the primitives do not exist in Ironic.
- **project - This is the logical grouping in which users are members of projects**
  and have some level of implied member rights with-in that project. This is tracked and associated using the `project_id` value.

Additional information can be found in the *Keystone administration - tokens <https://docs.openstack.org/keystone/latest/admin/tokens-overview.html#authorization-scopes>* documentation and the Keystone contributor - services documentation.

Problem description

The fundamental issue at hand is Ironic does not understand scoped access. This lack of understanding of scoped access coupled with existing project scoped role/access creates a confusing and different authentication experience from other services which have implemented the ability to have scope delineated access, being `Keystone` and `Nova` as of the point in which this specification was authored.

Coincidentally there is a desire from larger OpenStack operators to have the ability to delineate access. In other words permit operations centers to be able to view status, but not be able to act upon nodes. This may be a `reader` scoped user in the scope of a project, or in the scope of the entire system.

As projects within OpenStack implement Scope and Role delineation, and enable scope based access restriction, a risk exists that Ironic will become incompatible with the models attempting to be represented.

And thus we must implement support to delineate scopes, roles, and ultimately what may be a differing access model for some remote resources. In particular, risk exists with existing integrations as they may grow to expect only Project scoped requests, and refuse a System scoped member request. These sorts of issues will need to be identified and appropriately navigated.

In summary, the purpose of this specification is to make changes in *ironic* and *ironic-inspector* to be consistent and future compatible with the rest of the OpenStack community. This will further enable infrastructure operators where they can leverage the prior community policy work in the OpenStack community to override the policy defaults the community reaches.

Proposed change

At a high level, the desire is to:

a) Have greater consistency through the adoption of standard roles.

b) Implement the ability to move to the standard scope based restriction where the new standardized roles would apply.

c) Move services, such as ironic from the concept of *admin projects* to a *system scope*.

We will do this by:

1) Constructing a new set of policies to reflect the secure RBAC model where the scope is included as part of the definition.

2) Deprecating the previous policies in code which consist of roles scoped to the `baremetal` project. These should be anticipated to be removed at a later point in time.

3) Implementing explicit testing to ensure scopes are handled as we expect.

4) Creating an integration test job leveraging the `oslo.policy` setting to enforce scope restriction to help ensure cross-service compatibility and potentially having to alter some cross-service interactions to ensure requests are appropriately modeled. It should be expected that this may make visible any number of possible issues which will need to be addressed.

During the deprecation period, operators will continue to be able to leverage the previous authentication model.

These new policies would model our existing use and data model however with scope applied *and* multi-tenant access enabled. This will enable a friendly default usage path which will still be opt-in unless the node `owner` or `lessee` field is populated on a node object.

Combining the three defined roles of `admin`, `member`, and `reader`, with the three scopes, `system`, `domain`, `project` results in a matrix of possibilities. But, the `domain` is not anticipated to be needed, thus leaving six access scenarios or personas that have to be considered.

Please consult the *High level matrix* for a high level overview as to the anticipated use model.

In order to have a consistent use pattern moving forward, the existing role definitions of `baremetal_admin` and `baremetal_reader` will be deprecated and removed, however they will also not be effective once the `[oslo_policy]enforce_scope` and `[oslo_policy]enforce_new_defaults` parameters are set to `True`.

Above and beyond new policy definitions, the creation of additional tests will be needed in the `ironic` and `ironic-inspector` projects to validate enforcement or appropriate resource denial based upon the scope.

Additional issues and rights validation logic may need to be applied, however that will likely require adjacent/integrated projects to change their policy enforcement.

Note

Adjacent/integrated projects/services, for example is the interaction between Nova, Neutron, Cinder, Glance, Swift, and Ironic. Services do convey context on behalf of the original requester for a period of time, and can make access control decisions based up on this. Ironic has previously had to address these sorts of issues in the Neutron and Cinder integrations.

In terms of `ironic-inspector` and its API, the resulting default policies for this effort would be entirely system scoped and no other scope is anticipated to need implementation as the `ironic-inspector` is purely an admin-only and hardware data collection oriented service.

Note

In review of this specification document, it has been highlighted that a tenant may find it useful to have the ability to trigger inspection of a node, and have it report to *their* own `ironic-inspector` instance. This is an intriguing possibility, but would be a distinct feature above and beyond the scope of this specific work. The benefit of the previous policy in code effort, is operators should be able to simply update the policy in this case, if operationally permissible in that Operators security posture.

High level matrix

The table below utilizes two definitions which hail back to the existing multitenancy work that is present in ironic. They are not the proposed new name, but used to provide conceptual understanding of what the alignment of the policy rule represents since there are technically several different access matrices based upon the variation and ultimately the agreement reached within the community. The end name definition may be something similar, but that is an implementation naming decision, not higher level design decision.

- ***is_node_owner* - When the API consumers project ID value is populated in**
  the Ironic node objects `owner` field. This represents that they are the authoritative owner of the baremetal node.
- ***is_node_lessee* - When the API consumers project ID value is populated in**
  the Ironic node objects `lessee` field. This is considered the current or assigned user of the node. See the Allow Leasable Nodes specification for additional details.

Note

It is important to stress, that the table below are general guidelines. A higher level of detail is available below in *Project Scope* and *Endpoint Access Rights*.

Role	System Scope	Project Scope
admin	Effectively the same as the existing baremetal_admin role.	Project `admin` able to have equivalent access to the API as `system` scoped `member` with a filtered view matching *is_node_owner*. `owner` field updates are blocked. Some sensitive fields may be redacted or be restricted from update.
member	New concept for a *do-er* user or service account. Cant add or delete nodes, but can do things like provision_state changes.	Project members will be able to use a baremetal node if *is_node_lessee* or *is_node_owner* is matched and perform field/state updates on individual nodes with the exception of the `owner` and `lessee` fields. Some additional fields or update restrictions will exist.
reade	Effectively the same as the existing baremetal_observer	This is a read-only user concept where a project `reader` would be able to view a node if *is_node_owner* or *is_node_lessee* applies. This role is expected to still have a restricted view, which will likely vary based on which type of granted rights.

Note

An `auditor` role has not been proposed in this work, but *does* make eventual sense in the long term, and should be logically considered as reader does not equal an auditor in role. The concept for `auditor` would expect to allow secrets such as masked fields to be unmasked.

Note

Some role/scope combinations may be combined in discussions and communication in a {scope}-{role} format. This is effectively the persona being defined. Such as *system-admin* for a system wide scope or *project-admin* for a user who is a project administrator.

Note

Field restriction are likely to be controlled by additional policy rules, which MAY cascade in structure where if full general update access is not granted then lower level policies should be enumerated through. Similar logic is already present in ironic.

In effect, a `PROJECT_ADMIN`, if defined in the terms of a rule, would match upon a `project_id` matching the `owner` and the user having an admin role. A `PROJECT_MEMBER` includes `PROJECT_ADMIN` *or* where `project_id` matches `lessee` and the role is `member`.

Alternatives

No alternative is available as the model of implementation. This is due to it attempting to conform to the overall OpenStack model. Fine details should likely be discussed with-in the implementation.

Data model impact

None

State Machine Impact

None

REST API impact

The overall high level behavior of this change will be settings enforced through `oslo_policy` until the deprecated policies are removed from Ironic.

In accordance with API standards, even though it will not modify functional behavior this change will increment the API micro-version. This is to enable API consumers to be able to navigate around possible logic or policy changes around an upgrade. This is unrelated to policy enforcement specifics which cannot be permitted to be visible via the API surface.

End API user behavior is not anticipated to be changed, however with scope enforcement set in `oslo.policy`, an appropriately scoped user will be required.

System Scope

The transition for System scoped roles is fairly straight forward as described by the chart *High Level Matrix* in *Proposed Change*. Existing Admin/Observer roles would be translated to System-Admin and System-Reader respectively.

The addition to this scope is the `member` role concept. This is a user who can *Read* and *Update*, but that cannot *Create* or *Delete* records. In other words, the API consumer can deploy a node, they can update a node, but they are unable to remove a node. They should be able to attach/detach VIFs, and ultimately this should be able to be the rights granted to the service account used by the `nova-compute` process.

A user with a system scope of any valid role type should be anticipated as having full API surface visibility with exception of the special purpose `/v1/lookup` and `/v1/heartbeat` endpoints. This will be different for *Project Scope* based access where nodes will only be visible if owner or lessee are populated.

Todo

Follow-up with neutron regarding port attach/detach.

Todo

Follow-up with Cinder regarding volume attach/detach.

Todo

Follow-up with Nova regarding rights passed through on context.

Note

The primary focus of this specification is targeted at the Wallaby development cycle where the System scope is most beneficial to support. Given time constraints and cross-project mechanics we will likely see additional work to refine scope interactions under this spec as time progresses. Some of these things may be related to `volume` or `port` attachments, or possibly even tighter integration of this functionality in `nova-compute`. All of these things will evolve over time, and we cannot answer them until we reach that point in time.

Project Scope

The Project scoped restrictions in the secure RBAC model are dramatically different, however precedent already exists with the addition of the *is_node_owner* and *is_node_lessee* logic which would apply to project scoped interactions.

API consumers seeking to `GET` resources in the project scope would only be able to view resources which match the `is_node_owner` and/or `is_node_lessee` which are associated to the `owner` and `lessee` fields.

Note

A node CAN have an `owner` and/or `lessee` independently, and at present the policy model delineates access separately.

In this case, a Project-Admin would have similar rights to a System-Member where they would be able to update hardware focused fields such as `driver_info`, however only if `is_node_owner` matches. Project admins who match `is_node_lessee` should not be permitted the ability to update fields such as `driver_info`.

Todo

We may wish to evaluate if it is useful to permit updating `driver_info` as a project admin. Dtantsur thinks, and I agree that this is likely highly deployment and operationly specific, and it may be we need a knob to govern this behavior.

A Project-Member would again be scoped to the appropriate database entries which apply to their users scope. They should be enabled to update fields such as `instance_info`, and provision, unprovision, and potentially update VIFs.

VIFs being set will need to have some additional code to perform an access rights verification to ensure that a project member is attempting to bind to a VIF which matches their node ownership and their users entry, or the value of the lessee field and that requesting users project.

With the physical nature of assets, project scoped users are unable to create or delete any records.

Project scoped readers, again would only have a limited field view with the associated `is_node_lessee` or `is_node_owner`.

Endpoint Access Rights

This list is based upon the published information in the Baremetal API Reference. Not all actions on the node object are covered in this list. Some field restrictions apply. See *Node object field restrictions* for details with a Node object.

Note

This list does not include all possible actions on a node at this time.

Endpoint	Project Scope Accessible
/	Yes, Public endpoint
/v1	Yes, Public endpoint
/v1/nodes	Filtered View and access rights which will necessitate additional policy rules to be added.
/v1/nodes/{uuid}	Filtered view and access rights
/v1/nodes/{uuid}/vendor_pa	No, Will not be permitted as this is a open-ended vendor mechanism interface.
/v1/nodes/{uuid}/traits	Yes, accessible to `owner` to manage
/v1/nodes/{uuid}/vifs	Yes, write access requires additional validations.
/v1/portgroups	Yes, Filtered view and Read-Only for `owner` managability.
/v1/nodes/{uuid}/portgroups	Filtered view and Read-Only
/v1/ports	Yes, Filtered view and access rights for `owner` managability.
/v1/nodes/{uuid}/ports	Filtered view and access rights.
/v1/volume/connectors	Yes, Filtered view, Read-only.
/v1/volume/target	Filtered view, will require extra to prevent target requested is valid for the user/project to request.
/v1/nodes/{uuid}/volume/co	Filtered view, read-only.
/v1/nodes/{uuid}/volume/tar	Filtered view, read-only.
/v1/drivers	No, *system* scope only.
/v1/nodes/{uuid}/bios	Yes, Filtered view based on access rights to the underlying node.
/v1/conductors	No, *system* scope only.
/v1/allocations	Project scoped, however the access model is geared towards owners using this endpoint.
/v1/deploy_templates	No, *system* scope only at this time. as the table/data structure is not modeled for compatibility.
/v1/chassis	No, *system* scope only.
/v1/lookup	No, Agent reserved endpoint.
/v1/heartbeat	No, Agent reserved endpoint.

Warning

Port support will require removal of legacy neutron port attachment through `port.extra['vif_port_id']`

Note

Contributor consensus is that `port` objects do not require project scoped access, however one important item to stress is that the `owner` may be viewed as the ultimate `manager` of a physical node, and the `system`, or `ironic` itself just provides the management infrastructure. This is a valid case and thus it may be reasonable that we settle on permitting owner far more access rights than node lesses in a project scope.

Note

Contributor consensus is that resource class and trait records may only be necessary for a `system` scoped user to edit, however the case can also be made that this should be able to be delegated to the `owner`. This specification, itself, is not calling for a specific pattern, but more so anticipates this will be an implementation detail that will need to be sorted out. It may start as something only `system` scoped users with the appropriate role can edit, and may evolve, or it may not be needed.

Node object field restrictions

Note

These are proposed, however not final. Implementation of functionality will determine the final field behavior and access.

- uuid - Read-Only
- name - Read/Write for Project Admins if the project owns the physical machine.
- power_state - Read-Only
- target_power_state - Read-Only
- provision_state - Read-Only
- target_provision_state - Read-Only
- maintenance - Read/Write
- maintenance_reason - Read/Write
- fault - Read/Write
- last_error - ??? .. TODO:: The issue with `last_error` is that it can leak infrastructure hostnames of conductors, bmcs, etc. For BMaaS, it might make sense?
- reservation - Returned as a True/False for project users.
- driver - Read-Only
- driver_info - Likely returns as an empty dictionary, although alternatively we can strip the URLs out, but that seems a little more complicated.
- driver_internal_info - Likely will return an empty dictionary as Project Admins and Project Members should not really need to see the inner working details of the driver.
- properties - Read-Only

- instance_info - Project Admin/Project Member Read-Write
- instance_uuid - Read/Write for Project Admin/Project Member
- chassis_uuid - Returns None
- extra - Project Admin/Project Member Read-Write .. TODO:: another reason to remove old vif handling logic is the extra field.
- console_enabled - Project Admin/Project Member Read/Write
- raid_config - Read-Only
- target_raid_config - Read-Only
- clean_step - Read-Only
- deploy_step - Read-Only
- links - Read-Only
- ports - Read-Only
- portgroups - Read-Only
- resource_class - Read-Only
- boot_interface - Read-Only
- console_interface - Read-Only
- deploy_interface - Read-Only
- inspect_interface - Read-Only
- management_interface - Read-Only
- network_interface - Read-Only
- power_interface - Read-Only
- raid_interface - Read-Only
- rescue_interface - Read-Only
- storage_interface - Read-Only
- traits - Read-Only
- vendor_interface - Read-Only
- conductor_group - Returns None/Read-only
- protected - Read/Write
- protected_reason - Read/Write
- owner - Read-Only and lessee will be able to see the owner ID.
- lessee - Project Admin/Project Member Read-Write. Lessee will be forbidden from changing the field value.
- description - Read-Write
- conductor - Returns None as it provides insight into the running infrastructure configuration and state, i.e. System visible is the only appropriate state.

- allocation_uuid - Read Only

Special areas:

volume - This represents volume targets and connectors. All values
visible through this path should be read-only. Connector logic should be read/write accessible to Project Admins or Project members where applicable, however additional logic checks need to exist under the hood to validate permission access for the project and user.

state - This is the entry path towards changing state, indicators,
provisioning, etc. This should be permitted for Project Admin or Project Member IF it maps the associated owner or lessee field.

vendor_passthru - Vendor passthrough will not be available to project
scoped users in the RBAC model.

Note

All fields that are scrubbed, i.e. set to None or {} are expected to be read-only fields to project scoped accounts in the new RBAC model.

Client (CLI) impact

openstack baremetal CLI

None anticipated.

openstacksdk

None anticipated.

RPC API impact

At this time, no impact to the RPC API is anticipated. That being said the possibility does exist, given the nature of the security changes, some changes may be required should an additional argument be required. Existing patterns already exist for this and any such changes would be navigated with the existing rpc version maximum and pin capability.

Driver API impact

None.

Nova driver impact

We may wish to go ahead and establish the ability for nova to store the users project ID in the node `lessee` field. In the new use model, this would allow a more natural use pattern and allow users to be able to leverage aspects like power operations or reboot or possibly even rebuild of their deployed instances.

Todo

We should discuss this further. It likely just ought to be a knob for nova-compute with the Ironic virt driver.

Ramdisk impact

None anticipated as the existing heartbeat and lookup resources of the API would not be modified.

Security impact

The overall goal of the Secure RBAC work is to enable and allow an operator to be able to run a service in a more restrictive and constrained model where greater delineation exists between roles.

In a sense, the system scoped operating mode will eventually become the normal operating mode. This is in order to encourage more secure environments, however this will entirely depend upon the default policies *and* the policies operators put in place which may override the default policy. The overall goal of this specification also being to help identify the new policy mechanics.

In order to help manage this and ensure the overall behavior is enforced as expected, we anticipate we will need to create API behavior testing to ensure operational security and validate that future code changes do not adversely impact permission enforcement.

Other end user impact

No direct end-user impact is anticipated.

Scalability impact

None.

Performance Impact

No direct performance impact is anticipated. The object model already pushes the list filtering down to the DBAPI level, which is ideal for overall performance handling. It is likely some additional checks will produce a slight overhead, but overall it should be minimal and confined to logic in the API services.

Other deployer impact

Cloud infrastructure operators are anticipated to possibly need to adjust `oslo_policy` settings to enable or disable these new policies. This may include cloud operators continuing to use older or other more restrictive policies to improve operational security.

Developer impact

None anticipated at this time.

Implementation

Assignee(s)

Primary assignee:
Julia Kreger (TheJulia) <juliaashleykreger@gmail.com>

Other contributors:
Steve Baker (stevebaker) <sbaker@redhat.com>

Work Items

- Creation of positive/negative policy check tests that represent the current interaction models.
- Creation of scoped policy definitions and associated positive/negative behavior tests:
  - Creation/migration of such for System-Admin where the admin tests appropriately enforce that continity is the same for a scoped admin as with previous tests.
  - Creation/migration of such for System-Reader where values are visible but not able to be written to.
  - Creation of similar for System-Member
  - Creation of similar for Project-Admin
  - Creation of similar for Project-Member
  - Creation of similar for Project-Reader
- Implementation of a CI job which operates a full integration sequence *with* scope policy enforcement enabled via the `[oslo_policy]` configuration.
- Documentation!

Phases

The initial phase for deployment is scoped for the eqiuvalent of the existing project admin scoped authentication for system scoped use.

The next phase, presumably spanning a major release would then cover the project scoped access rights and changes.

Dependencies

Minimum versions of `oslo_policy` will need to be updated to match the Victoria development cycles work product, however this is anticipated to be completed as part of the JSON to YAML policy migration effort.

Testing

An CI integration job is anticipated and should be created or one already leveraged which is utilising the widest configuration of integrated components to ensure that policies are enforced and this enforcement works across components. Due to the nature and scope of this effort, it may be that Ironic alone is first setup to scope limit authorizations as other projects also work in this direction.

Upgrades and Backwards Compatibility

Not applicable.

Documentation Impact

Release note will need to be published with the prior policy deprecation as well as primary documentation updated to reflect the scope based configuration. An in-line documentation warning will likely be necessary depending on what the larger community decides in terms of the RBAC policy efforts and end-user/operator needs to be.

References

- https://review.opendev.org/c/openstack/ironic/+/763255
- https://review.opendev.org/q/topic:%2522secure-rbac%2522+(status:open+OR+status:merged)+project:openstack/ironic
- http://lists.openstack.org/pipermail/openstack-discuss/2020-November/018800.html

5.15.192 Modifying Active Nodes with Service Steps

https://storyboard.openstack.org/#!/story/2010647

A reality in the operation of systems is that sometimes things need to be changed. An additional unfortunate reality is that the closer we get to the underlying infrastructure, the greater the burden it is to take a model of just redeploy with new settings.

Unfortunately, Ironic has limited options in this area for infrastructure operators who wish to automate activities such as upgrading firmware, or changing low level settings when a system is already deployed. Today they can possibly leverage the **rescue** process, but that also means they have to articulate everything after the rescue process had completed.

Yet Ironic also has quite a bit of tooling to enable settings to be changed as well as firmware upgraded as part of cleaning and deployment, and we should provide capability to leverage this tooling on a node in an **active** state.

Problem description

Deployed systems need to be able to be changed or evaluated:

- Firmware settings and software sometimes need to be modified or updated.
- The introduction of data processing units (the further evolution of SmartNICs) with full operating systems, means additional maintenance actions can arise, and direct access to the device may not be feasible.
- Security Operations teams sometimes need to be able to inspect the state of the hardware and software from an out of bands process, which may result in the hardware being removed from inventory, or returned to a running state if no issue are detected.
- Operations teams needing to test the Memory, CPU, or Network in order to identify an issue or address a customer concern.

In these cases, it may be that hardware managers and/or vendor driver methods can facilitate some of the operator required actions. In other cases, it may require additional tools that may not be suitable to install under normal conditions.

Proposed change

The implementation of a capability to take a node in an **active** state, execute steps and return the node to an **active** state, as well as a standing state to allow operators to perform their other needful actions.

To do this, we will implement a **service** API verb which will move a node into a **servicing** state, in line with the existing cleaning and deploy processes. The IPA ramdisk will be booted, and the node will enter a **service wait** state.

Once IPA is online, two possible paths can be taken.

If the **service** verb was invoked with with a `service_steps` parameter, in the same line of `clean steps`, then the node state will return to the **servicing** state, heartbeat, and upon all steps completed, the node will automatically return to the **active** state.

To achieve this, we will also need to:

- Add an additional `service_step` field to the Node object. Similar to clean steps, a `service_steps` entry would also be utilized in the node `driver_internal_info` field.
- Increment the API version.
- Introduction of a `unhold` provision state verb to allow resumption of step processing/execution.
- Make appropriate decorator modifications in alignment with deploy steps and cleaning steps, which would also be in line with existing deployment and cleaning steps.
- Introduction of, or modification of existing housekeeping periodic steps to include handling for the newly proposed states. In particular, modification is preferred from a database efficiency standpoint, however it is ultimately the implementers prerogative.

In terms of ramdisk usage, the existing `deploy_kernel` and `deploy_ramdisk` will be used by default for this feature, however a `service_kernel` and `service_ramdisk` will also be available as an override.

To facilitate a handoff or pause/resumption operation, we will introduce two explicit steps into the step process.

- `hold` - This step would hold execution of steps on the current node in its current state unless an `unhold` command has been issued, or a new set of steps have been received.
- `pause` - This step would pause the step execution for a user defined period of time. Think of this as sleep. In all likelihood, this step is likely to be constrained with a maximum sleep period of the heartbeat window.
- `wait` - This step would allow an operator tell Ironic to wait for a condition to be met before proceeding with steps. This may be a file to appear on the agent, or for a agent local command to execute successfully. This step is likely to actively await a true condition to be met, which for logical reason would allow automatic resumption. For example an async firmware step which needs to be waited upon.

These temporary holding states, when used, will take a case appropriate path, such as with hold, the node will move to a `clean hold`, `deploy hold`, or in this specifications case, `service hold` state. Pause and wait, actions will be situational and will likely be treated as transitory within to be determined constraints.

As for hold, a conductor failover is not anticipated to be fatal to this in the long term, as long as the agent continues to heartbeat. An `unhold` provision_state action is expected to just remove the current step name in progress, allowing the next step to proceed upon next heartbeat.

Note

Ironic *might* need a step which enables the agent token to be removed to allow the agent to be restarted or rebooted. This is not anticipated to be a feature, but shouldnt be controversial if deemed needed soon afterwards.

Alternatives

One possible alternative appears to be continue and encourage the use of the rescue framework, which is not ideal and has no capability for step execution, meaning any action taken would need to be manually composed and executed externally. Of course, this is possible today to partly achieve the same basic effect. Rescue itself helped inspire this overall specification as a superset of functionality.

Another possibility could be to extend the existing rebuild verb to allow steps to be passed in a non-destructive, non-redeploy scenario. However the rebuild verb has always suggested re-deployment, and separating that identity might prove difficult and time consuming to deliver only part of the needful functionality. Contributors are not convinced this is even a viable alternative and have also noted concerns of operator confusion, resulting in this being a further unlikely alternative.

Data model impact

Addition of a new `node` field `service_step`, which will require a new column to be added to the database, however that is a relatively low impact database change. This field will not be indexed.

Use of subfield values **driver_internal_info[service_steps]** and **driver_internal_info[service_step_index]**.

State Machine Impact

New states will be added to the state machine:

State	Description
states.SER	Modifying unstable state indicating a lock is held and action is occurring.
states.SER	An intermediate unstable state where Ironic is waiting for an action such as a *heartbeat* from the agent to begin. begin.
states.SER	An error state to indicate there was an error in the process of handling the request. This is a stable state until the operator removes the node from it. This could be a result of any failure in any service or unservice process.
states.SER	A stable state for nodes being held in a state based upon use of the `hold` step. Removal of the node from the hold state would involve the `unhold` provision state verb.

The general flow will be:

> **ACTIVE -> SERVICING -> SERVICEWAIT -> SERVICE**

In the case of an automated flow:

> **ACTIVE -> SERVICING -> SERVICEWAIT -> SERVICING ->**

In the event that the a user determines they need to stage actions, a service step should be able to be called while already in the service state.

> **SERVICE -> SERVICING -> SERVICEWAIT -> SERVICE**

In the caes of entirely conductor side modifications, such as Out-of-Band firmware updates being applied:

> **ACTIVE -> SERVICING -> ACTIVE**

In the event of an error, the operation can be retired or the node returned to an *active* state:

> **SERVICING -> SERVICEFAIL -> ACTIVE SERVICING -> SERVICEFAIL -> SERVICING**

To facilitate the workflow enhancements related to this, additional states will be added for alignment with existing step framework changes.

State	Description
states.DEP	A hold state to allow use for the `hold` step name to allow cross-step framework capabilities.
states.CLE	An identical state with different state name to the previously indicated `deploy hold` state. The delineated names in large part to prevent confusion and the need for additional logic for `target_provision_state`.

In addition to these new states, new state verbs are anticipated:

- service - The verb to trigger service steps framework.
- unhold - The verb to trigger release of a holding state and enable steps to continue to execute.

In the process of moving back from a service state, the node will have a boot to the default boot device if the ramdisk is booted in the process of of executing the service request.

REST API impact

In line with existing community practice in regards to the nodes provision state field, the contents of the `provision_state` field will not be version guarded.

Note

While this seems like a potentially breaking change, we have only done this when weve renamed or changed the overall meaning of a state. i.e. None -> Available and Inspect Wait -> Inspecting for older API clients. In other words, these being net-new should not be breaking.

Note

Nova impact is covered below.

The ability to submit the new provision state verbs to the API will be version guarded, and the payload of the new `service_steps` will align with existing `clean_steps` and `deploy_steps` through use of a JSON payload. Ability to process the request will also be dependent upon the RPC version in use.

A corresponding API client change will also be required, and an RBAC policy will also be added to allow for restriction of the service verb, as project scoped `lessee-admin` users are able to issue provision state changes. The access scope for the rule is anticipated to be restricted to appropirate system scoped and project scoped owner roles, in line with the existing policy alias of `SYSTEM_MEMBER_OR_OWNER_ADMIN`.

Client (CLI) impact

openstack baremetal CLI

Addition of a `baremetal node service` and `baremetal node unservice` commands will need to be added to the client.

openstacksdk

The `set_provision_state` method in OpenStackSDKs will need an additional argument to enable passing through service steps.

RPC API impact

A new `do_node_service` RPC method will be required which will also require the RPC interface version to be incremented.

The act of removing a node from the service state is anticipated to use the `do_provision_action` RPC method, but the API may need to validate the running RPC version before making the call.

Because of this addition, both services will need to be upgraded before this feature can be utilized.

Driver API impact

Decorators will need to be added/modified to enable the steps to be appropriately validated and invoked when the feature is needed. No other driver API changes are anticipated. It should be noted this is not a breaking change, we only note it because it will need to be performed on relevant actions/steps.

Nova driver impact

A review of Novas virt/ironic/driver.py code suggests that no impact is to be anticipated through the introduction of new states as they are ignored. As such, considering our practice of not concealing new states behind API version changes when not impacting to Nova, we believe no additional handling will be needed.

We *may* want to introduce a version guard to return a VirtDriverNotReady exception should an action such as unprovisioning or vif actions are attempted by a Nova user while the baremetal node is in one of these states, as this is a generally non-fatal not ready at the moment exception, but that should be further discussed with the Nova team.

Ramdisk impact

A `get_service_steps` and `execute_service_step` methods are anticipated as being needed to support this functionality in the agent. A lack of these agent side commands are not to be considered faults.

It is expected that we would likely want the agent to just to continue to heartbeat while waiting for work to do, which is not breaking for older versions of the agent.

Security impact

A potential security impact exists in that under Ironics default security model, a lessee admin is able to trigger provision state changes. It *is* an operationally valid use case to permit a lessee, at least a lessee who was manually assigned, to be able to service firmware as long as it is inline with approved and known versions.

It might be best to drive forward keylime integration as well, as well as place a policy rule on use of the service verb.

Note

We may wish to consider permitting the agent to heartbeat and the callback URL if it has been moved to a different network. This does have a security impact, but would allow fast-track to be applied on a more consistent basis.

Other end user impact

None

Scalability impact

To reconcile errors, it may be necessary to introduce an additional periodic task in order to identify failures. Beyond this potential additional periodic state, no scalability impact is anticipated.

Performance Impact

No performance impact is anticipated.

Other deployer impact

This feature acknowledges and encourages operators to integrate in ways that recognize we may not be the driver of the workflow, but that the node needs to be put into a particular state before proceeding.

With that being said, no direct deployer impact is anticipated, but operators will likely need and anticipate quite a bit of information to appropriately explain the feature, and how they can leverage it to both automate their workflows and have a consistent operational experience.

Developer impact

None anticipated.

Implementation

Assignee(s)

Todo

Volunteers? Happy to hack on this if I can get people to commit to review.

Primary assignee:
<IRC handle, email address, or None>

Other contributors:
<IRC handle, email address, None>

Work Items

- Add `service_step` to the node object model and database.
- Update the state machine configuration
- Add step retrieval methods to the agent.
- Add conductor internal methods for triggering state actions and calls.

- Add agent client method call to retrieve a list of steps from the agent.
- Add an agent client method call to re-trigger DHCP of the agent as a service step as well.
- Add RPC method for service action.
- Add API support for service verb.
- Add networking internals to place the machine on to a valid network for the service operations.
- Decorate appropriate step actions as service steps.
- Compose tempest test schenario.

Dependencies

None

Testing

A tempest scenario test for this will be needed. It is expected that a test scenario will exercuse one of the `pause`, `wait`, or `hold` steps.

Upgrades and Backwards Compatibility

No additional upgrade or compatibility issues are anticipated aside from what has already been noted and explored in this document.

Documentation Impact

Our admin documentation will need an additional section added to cover this feature, much as other major features have needed in the past.

References

- https://specs.openstack.org/openstack/ironic-specs/specs/11.1/deployment-steps-framework.html
- https://specs.openstack.org/openstack/ironic-specs/specs/16.0/in-band-deploy-steps.html
- https://specs.openstack.org/openstack/ironic-specs/specs/10.1/implement-rescue-mode.html

5.15.193 Shard Key Introduction

https://storyboard.openstack.org/#!/story/2010378

After much discussion and attempts to remedy the scalability issues with `nova-compute` and its connection to Ironic in large scale deployments, and upon newly discovered indicators of `networking-baremetal` having a similar scaling issue, the community has started to reach an agreement on a path forward. Specifically, to introduce a sharding model which would allow API consumers to map and lock on to specific sets of baremetal nodes, regardless of if the relationship is semi-permanant or entirely situational. Only the consumer of the information performing processing can make that determination, and it is up to Ironic to try and provide the substrate capabilities to efficiently operate against its API.

Problem description

The reality is Ironic can be used at some absurd scales in the hundreds of of thousands of baremetal nodes, and while *most* operators of Ironic either run multiple smaller distinct Ironic deployments with less than 500 physical machines, some need a single deployment with thousands or tens of thousands of physical nodes. At increased scales, external operations polling ironic, generally struggle to scale at these levels. It is also easy for misconfigurations to be made where performance can become degraded, which is because the scaling model and limits are difficult to understand.

This is observable with the operation of Novas Compute process when running the `nova.virt.ironic` driver. It is operationally easy to get into situations where one is attempting to support thousands of baremetal nodes, with too few `nova-compute` processes. This specific situation leads to the process attempting to take on more work than it was designed to handle.

Recently we discovered a case, while rooted in misconfiguration, where the same basic scaling issue exists with `networking-baremetal` where it is responsible for polling and updating physical network mappings in Neutron. The same basic case, a huge amount of work, and multiple processes. In this specific case, multiple (3) Neutron services were stressing the Ironic API retrieving all of the nodes, and attempting to update all of the related physical network mapping records in neutron, resulting in the same record being updated 3 times, once from each service.

The root issue is the software consuming Ironics data needs to be able to self-delineate the overall node set and determine the local separation points for sharding the nodes. The delineation is required because the processes executed are far more processor intensive, which can introduce latency and lag which can lead towards race conditions.

The challenge, from what has been done previously, is the previous model required downloading the entire data set to build a hash ring from.

Where things are also complicated, is Ironic has an operational model of a `conductor_group`, which is intended to help model a physical grouping or operational constraint. The challenge here is that conductor groups are not automatic in any way, shape, or form. As a result, conductor groups is not the solution we need here.

Proposed change

Overall the idea, is to introduce a `shard` field on the node object, which an API user (Service), can utilize to retrieve a subset of nodes.

This new field on the node object would be inline with existing API field behavior constraints and can be set via the API.

We *can* provide a means to pre-set the shard, but ultimately it is still optional for Ironic, and the shard *exists* for the API consumers benefit.

In order to facilitate the usage by an API client, `/v1/nodes` and `/v1/ports` would be updated to accept a `shard` parameter (i.e. GET /v1/nodes?shard=foo, GET /v1/ports?shard=foo, GET /v1/portgroups?shard=foo) in the query to allow for API consumers to automatically scope limit their data set and self determine how to reduce the workset. For example, `networking-baremetal` may not care about assignment, it just needs to reduce the localized workset. Whereas, `nova-compute` needs the shard field to remain static, that is unless `nova-compute` or some other API consumer were to request the `shard` to be updated on a node.

Note

The overall process consumers use today is to retrieve everything and then limit the scope of work based upon contents of the result set. This results in a large overhead of work and increased looping latency which also can encourage race conditions. Both `nova-compute` and the `networking-baremetal` ML2 plugin operate in this way with different patterns of use. The advantage of the the proposed solution is to enable the scope limiting/grouping into manageable chunks.

In terms of access controls, we would also add a new RBAC policy to restrict changes such that the system itself or a appropriately scoped (i.e. administrative) user can change the field.

In this model, conductors do not care about the shard key. It is only a data storage field on the node. Lookups for contents of the overall shard composition/layout, for GET /v1/shards, is to be performed directly against the nodes table using a SQL query.

Alternatives

This is a complex solution to allow simplified yet delineated usage, and there are numerous other options for specific details.

Ultimately, each item should be discussed, and considered.

One key aspect, which has been recognized thus far, is that existing mechanisms can be inefficiently leveraged to achieve this. An example of this is that `conductor_group`, `owner`, `lessee` all allow for filtering of the node result set. A `conductor_group` being an explicit aspect an API client can request, where as `owner` and `lessee` are access control based filters tied to the API clients submitted Project ID used for Authentication. More information on why `conductor_group` is problematic is further on in this document.

Consensus in discussion with the Nova teams seems to be that usage of the other fields, while in part may be useful, and possibly even preferred in some limited and specific cases, doesnt solve the general need to be able to allow clients to self delineate *without* first downloading the *entire* node list first. Which in itself, the act of retrieving a complete list of nodes is a known scaling challenge, and creates increased processing latency.

In the `conductor_group` case, there is no current way to discover the conductor groups. Where as for `owner` and `lessee`, these are specific project ID value fields.

Why not Conductor Group?

It is important to stress similarity wise, this *is* similar to conductor groups, however conductor groups were primarily purposed to model the physical constraints and structure of the baremetal infrastructure.

For example, if you have a set of conductors in Europe, and a set of conductors in New York, you dont want to try and run a deploy for servers in New York, from Europe. Part of the attractiveness for this to be exposed or used in Nova, was *also* to align the physical structure. The immediately recognized bonus to operators was the list of nodes was limited to the running `nova-compute` process, if so configured. It is known to the Ironic community that some infrastructure operators *have* utilized this setting and field to facilitate scaling of their `nova-compute` infrastructure, however these operators have also encountered issues with this use pattern as well that we hope to avoid with a shard key implementation.

Where the needs are different with this effort and the pre-existing conductor groups, is that conductor groups are part of the hash ring modeling behavior where as in the shards model conductors will operate without consideration of the shard key value. We need disjointed modeling to support API consumer centric usage so they can operate in logical units with distinct selections of work. Consumers *may* also care about the `conductor_group` in addition to the shard because needing to geographically delineate

is separate from needing smaller chunks of work, or in this case groups of baremetal nodes for which a running process is responsible for.

In this specific case, `conductor_group` is entirely a manually managed aspect, which nova has a separate setting name due to name perception reasons, and our hope ultimately is something that is both simple and smart.

Note

The Nova project has agreed during Project Teams Gathering meetings to deprecate the `peer_list` parameter they forced use of previously to support conductor groups with the hash ring logic.

On top of this, Todays `conductor_group` functionality is reliant upon the hash ring model of use, which is something the Nova team wants to see removed from the Nova codebase in the next several development cycles. Where as, Ironic will continue to use the hash ring functionality for managing our conductors operating state as it is also modeled for conductors to manage thousands of nodes. These thousands of nodes just does not scale well into `nova-compute` services.

Why not owner or lessee?

With the RBAC model improvements which have taken place over the last few years, it *is* entirely possible to manage separate projects and credentials for a `nova-compute` to exist and operate within. The challenge here is management of additional credentials and the mappings/interactions.

It might be feasible to do the same for scaling `networking-baremetal` interactions with Ironics API, but the overhead and self management of node groupings seems onerous and error prone.

Also, if this was a path taken, it would also be administratively prohibitive for nova-computes nodes, and they would be locked to the manual settings.

What if we just let the API consumer figure it out?

This could be an option, but it would lead to worse performance and the user experience being worse.

The base conundrum is to orderly and efficiently enumerate through, and then acting upon each and every node API client is responsible for interacting with.

Today, Novas Compute service enumerates through every node, using a list generated upon one query, and it gets *most* of the data it needs to track/interact with a node, keeping the more costly single node requests to a minimum. If that client had to track things, it would still have to pull a full list, and then it would have to reconcile, track, and map individual nodes. Weve already seen this as not working using a Hashring today.

Similarly, `networking-baremetal` lists all ports. That is all it needs, but it has no concept of smaller chunking, blocks, or even enough information *to* really make a hashring which would represent existing models. To just expect the client to figure it out and to deal with that complexity, also means logic far away from a database. And for performance, the closer we can keep logic and decisions to an indexed column, the better and more performant, which is why the proposed solution has come forth.

Data model impact

Node: Addition of a `shard` column/value string field, indexed,
with a default value of None. This field is considered to be case sensitive, which is inline with the DB storage type. API queries would seek exact field value matches.

Note

We will need to confer with the Nova team and the nova.virt.ironic driver query pattern, to ensure we cover any compound indexes, if needed.

To facilitate this, database migrations, and data model sanity checking will need to be added to `ironic-status` as part of the upgrade checks.

State Machine Impact

None

REST API impact

PATCH /v1/nodes/<node>

In order to set a shard value, a user will need to patch the field. This is canned functionality of the existing nodes controller, and will be API version and RBAC policy guarded in order to prevent inappropriate changes to the field once set. Like all other fields, this operation takes the shape of a JSON Patch.

GET /v1/nodes?shard=VALUE,VALUE2,VALUE3

Returns a subset of nodes limited by shard key. In this specific case we will also allow a string value of none, None or null to be utilized to retrieve a list of nodes which do *not* have a shard key set. Logic to handle that would be in the DB API layer.

GET /v1/ports?shard=VALUE,VALUE2,VALUEZ GET /v1/portgroupss?shard=VALUE,VALUE2,VALUEZ

Returns a subset of ports, limited by the shard key, or list of keys provided by the caller. Specifically would utilize a joined query to the database to facilitate it.

GET /v1/shards

Returns a JSON representing the shard keys and counts of nodes utilizing the shard.

> {{Name: Shard-10, Count: 352}, {Name: Shard-11, Count: 351}, {Name: Shard-12, Count: 35}, {Name: null, Count: 921}}

Visibility wise, the new capabilities will be restricted by API micro-version. Access wise this field would be restricted in use to `system-reader`, `project-admin`, and future `service` roles by default. A specific RBAC policy would be added for access to this endpoint.

Note

The /v1/shards endpoint will be read only.

Client (CLI) impact

Typically, but not always, if there are any REST API changes, there are corresponding changes to python-ironicclient. If so, what does the user interface look like. If not, describe why there are REST API changes but no changes to the client.

openstack baremetal CLI

A `baremetal shard list` command would be added.

A `baremetal node list --shard <shard>` capability would be added to list all nodes in a shard.

A `--shard` node level parameter for `baremetal node set` would also be added.

A `baremetal port list --shard <shard>` capability would be added to limit the related ports to nodes in a shard. Similarly, the `baremetal portgroup list --shard <shard>` would be updated as well.

openstacksdk

A SDK method would be added to get a shard list, and existing list methods would be checked to ensure we can query by shard.

RPC API impact

None anticipated at this time.

Driver API impact

None

Nova driver impact

A separate specification document is being proposed for the Nova project to help identify *and* navigate the overall change.

That being said, no direct negative impact is anticipated.

The overall discussion revolving with Nova is to both facilitate a minimal impact migration, and not force invasive and breaking changes, which may not be realistically needed by the operators.

Note

An overall migration path is envisioned, but what is noted here is only a suggestion and how we perceive the overall process.

Anticipated initial Nova migration steps:

Ironic itself will not be providing an explicit process for setting the shard value on each node, aside from `baremetal node set`. Below is what *we, Ironic* anticipate as the migration steps overall to move towards this model.

1) Complete the Ironic migration. Upon completion, executing the database status check (i.e. `ironic-status upgrade check`) should detect and warn *if* a `shard` key is present on nodes in the database, but nodes exist without a `shard` value are present in the database.

2) The nova-compute service being upgraded is shut down.

3) A nova-manage command would be executed to reassign nodes to a user supplied `shard` value to match. Example: nova-manage ironic-reassign <shard-key> <compute-hostname>

   Programmatically, this would retrieve a list of nodes matching the key from Ironic, and then change the associated ComputeNode and Instance tables host fields to be the supplied compute hostname, to match an existing nova compute service.

   Note

   The command likely needs to match/validate that this is/was a compute hostname.

   Todo

   As a final step before the nova-manage command exits, ideally it would double check the state of records in those tables to indicate if there are other nodes the named Compute hostname is responsible for. The last compute hostname in the environment should not generate any warning, any warning would be indicitive of a lost ComputeNode, Instance, or Baremetal node record.

4) The nova-compute.conf file for the upgraded `nova-compute` service is restarted with a `my_shard` (or other appropriate parameter) which signals to the `nova.virt.ironic` driver code to not utilize the hash ring, and to utilize the blend of what it thinks it is responsible for from the database *and* what matches the Ironic baremetal node inventory when queried for matching the configured shard key value.

5) As additional compute nodes are migrated to using the new shard key setup, existing compute node imbalance should be settled in terms of the internal compute-node logic to retrieve what each node it thinks it is responsible for, and would eventually match the shard key.

This would facilitate an ability to perform a rolling, yet isolated outage impact as the new nova-compute configuration is coming online, and also allows for a flow which should be able to be automated for larger operators.

The manageability, say if one needs to change a `shard` or rebalance shards, is not yet clear. The current discussion in the Nova project is that rebalance/reassociation will only be permitted *IF* the compute service has been forced down which is an irreversible action

Ramdisk impact

None

Security impact

The `shard` key would be API user settable, as long as sufficient API access exists in the RBAC model.

The `/v/shards` endpoint would also be restricted based upon the RBAC model.

No other security impacts are anticipated.

Other end user impact

None Anticipated

Scalability impact

This model is anticipated to allow users of data stored in Ironic to be more scalable. No impacts to Ironics scalability are generally anticipated.

Performance Impact

No realistic impact is anticipated. While another field is being added, initial prototyping benchmarks have yielded highly performant response times for large sets (10,000) baremetal nodes.

Other deployer impact

It *is* recognized that operators *may* wish to auto-assign or auto-shard the node set programmatically. The agreed upon limitation amongst Ironic contributors is that we (Ironic) would not automatically create *new* shards in the future. Creation of new shards would be driven by the operator by setting a new shard key on any given node.

This may require a new configuration option to control this logic, but the logic overall is not viewed as a blocking aspect to the more critical need of being able to assign a node to a shard. This logic may be added later on, we will just endeveour to have updated documentation to explain the appropriate usage and options.

Developer impact

None anticipated

Implementation

Assignee(s)

Primary assignee:
Jay Faulkner (JayF)

Other contributors:
Julia Kreger (TheJulia)

Work Items

- Propose nova spec for the use of the keys (https://review.opendev.org/c/openstack/nova-specs/+/862833)
- Create database schema/upgrades/models.
- Update Object layer for the `Node` and `Port` objects in order to permit both objects to be queried by `shard`.
- Add query by shard capability to the Nodes and Ports database tables.
- Expose `shard` on the node API, with an incremented microversion *and* implement a new RBAC policy which restricts the ability to change the `shard` value

- Add pre-upgrade status check to warn if there are fields which are not consistently populated. i.e. `shard` is not populated on all nodes. This will provide visibility into the mixed and possibly misconfigured operational state for future upgrader.
- Update OpenStack SDK and python-ironicclient

Dependencies

This specification is loosely dependent upon Nova accepting a plan for use of the sharding model of data. At present, it is the Ironic teams understanding that it is acceptable to Nova, and Ironic needs to merge this spec and related code to support this feature before Nova will permit the Nova spec to be merged.

Testing

Unit testing is expected for all the basic components and operations added to Ironic to support this functionality.

We may be able to add some tempest testing for the API field and access interactions.

Upgrades and Backwards Compatibility

To be determined. We anticipate that the standard upgrade process would apply and that there would not realistically be an explicit downgrade compatibility process, but this capability and functionality is largely for external consumption, and details there are yet to be determined.

Documentation Impact

Admin documentation would need to include an document covering sharding, internal mechanics, and usage.

References

PTG Notes: https://etherpad.opendev.org/p/nova-antelope-ptg Bug: https://launchpad.net/bugs/1730834 Bug: https://launchpad.net/bugs/1825876 Related Bug: https://launchpad.net/bugs/1853009

5.15.194 SIGHUP on ironic services

https://bugs.launchpad.net/ironic/+bug/1585595

It is a common practice for daemons to use SIGHUP as a signal to reconfigure a server by reloading configuration files (as mentioned in Wikipedia). This specification describes adding this feature to the ironic-conductor and ironic-api services.

Problem description

Issuing a SIGHUP signal (via, for example, `kill -s 1 <process-id>`) to an ironic (ironic-api and ironic-conductor) service causes the service to be restarted.

It is a common practice for daemons to use SIGHUP as a signal to reconfigure a server by reloading configuration files (as mentioned in Wikipedia). This specification describes how ironic will support SIGHUP, such that select configuration options can be reloaded during runtime.

This is useful, for example, during a rolling upgrade. After unpinning (resetting the `pin_release_version` configuration option), SIGHUP could be used to restart the services with the updated option value.

Another example is for operators to more easily enable or disable the debug logging without having to stop and (re)start the services.

Note that SIGHUP is not supported on Windows.

Proposed change

We will leverage code from these libraries:

- the oslo.service library provides support for services to handle this signal. We would change the ironic services so that they are launched with `restart_method='mutate'`. When the library code handles the SIGHUP signal, it gets any changes for mutable configuration options.
- The oslo.config library adds support for `mutable` configuration options. Only mutable configuration options can be reloaded. A mutable option note (Note: This option can be changed without restarting.) is added to the description of a mutable configuration option in the ironic.conf.sample file. It logs any changed mutable configuration options. It also logs a warning for any changed options that are not mutable.

With these changes, when a SIGHUP occurs, the service will reload with values from the mutable options. A warning is logged for changes to any immutable options.

Mutable configuration options

Mutable configuration options that will be available are:

- options from other libraries. To date, the only mutable options that are used by the ironic services are from the oslo.log library:
  - `[DEFAULT]/debug`
  - `[DEFAULT]/log_config_append`
- `[DEFAULT]/pin_release_version`

Other ironic configuration options can be made mutable in the future; such changes should have corresponding release notes. The belief is that most, if not all, of the configuration options should be made mutable. However, that is outside the scope of this specification which is to lay the groundwork to make this possible with a small number of options. When mentioning that ironic supports SIGHUP, operators might assume (incorrectly) that this applies to all configuration options, so we should make other configuration options available in a timely fashion.

The value of a mutable configuration option should not be cached; or at least, if it is cached, the value must be updated upon a SIGHUP occurrence.

Alternatives

Change the desired configuration option values, stop the service, and then start it again.

Data model impact

None.

State Machine Impact

None.

REST API impact

None.

Client (CLI) impact

None.

ironic CLI

None.

openstack baremetal CLI

None.

RPC API impact

None.

Driver API impact

None.

Nova driver impact

None.

Ramdisk impact

None.

Security impact

None.

Other end user impact

The operator will be able to change certain configuration options and issue a SIGHUP to have an ironic service restart using the changed option values.

Scalability impact

None.

Performance Impact

None.

Other deployer impact

Among other things, this can be used for rolling upgrades.

Developer impact

None.

Implementation

Assignee(s)

Primary assignee: rloo (Ruby Loo)

Work Items

- change our services so they are launched with `restart_method='mutate'`
- change the desired configuration options so that they are mutable
- make sure the mutable options are not cached, or if they are, make sure that they are updated appropriately with a SIGHUP occurrence

Dependencies

None.

Testing

If we stop and restart a service in the e.g. multinode grenade testing, we could change that and issue a SIGHUP instead.

Upgrades and Backwards Compatibility

Changing the value of mutable configuration options will now take effect when a SIGHUP is issued. We need this to support rolling upgrades.

Documentation Impact

The use of SIGHUP (in the context of `[DEFAULT]/pin_release_version`) will be documented as part of the rolling upgrade process.

References

- Wikipedia
- oslo.service library
- oslo.config library
- rolling upgrade

5.15.195 The evolution of the Smart NICs to DPUs

https://bugs.launchpad.net/ironic/+bug/2019043

The ideas behind Smart NICs have evolved as time has progressed.

And honestly it feels like we in the Ironic community helped drive some of that improvement in better, more secure directions. Hey! We said we were changing the world!

What started out as highly advanced network cards which infrastructure operators desired to offload some traffic, have morphed into a more generic workload, yet still in oriented in the direction of offload. Except now these newer generations of cards have their own BMC modules, and a limited subset of hardware management can occur.

But the access model and use/interaction also means that a server can have a primary BMC, and then N number of subset BMCs, some of which may, or may not be able to communicate with the overall BMC and operating system.

And in order to really support this, and the varying workflows, we need to consider some major changes to the overall model of interaction and support. This is not because the device is just a subset, but a generalized computer inside of a computer with its own unique needs for management protocols, boot capabilities/devices, architecture, and has its own console, internal state, credentials, et cetra.

Problem description

Context Required

In order to navigate this topic, we need to ensure we have context of various terms in use and as they relate.

Smart NIC

These are best viewed as a first generation of DPU cards where an offload workload is able to be executed on a card, such as a Neutron agent connected to the message bus in order to bind ports to the physical machine.

Some initial community and vendor discussions also centered around further potential use cases of providing storage attachments through, similar to the behavior of a Fibre Channel HBA.

Composible Hardware

The phrase Composible Hardware is unfortunately overloaded. This is best described as use of a centralized service to compose hardware for use by a workload. A good way to view this, at least in a classical sense is through an API or application constructing a cohesive functioning computer resource with user definable CPU, Memory, Storage, and Networking. Essentially to virtualize the hardware interaction/modeling like we have with Virtual Machines.

Aside from some limited hardware offerings from specific vendors, Composible Hardware largely hasnt been realized as initially pitched by the hardware manufacters.

DPUs

A DPU or Data Processing Unit is best viewed as a more generalized, second generation of the Smart NIC which is designed to run more generalized workloads, however this is not exclusively a network or adapter to network attached resources. For example, one may want to operate a BGP daemon inside of the card, which is entirely out of scope ironic to operate with and manage, but they could run the service

there in order to offload the need to run it on the main CPUs of the system. A popular, further idea, is to utilize the card as a root of trust

A similarity between DPUs and Composible hardware in modeling is the concept of providing potentially remote resources to the running operating system.

Given the overall general purpose capabilities of DPUs and increased focus of specific computing workload offloads, we need to be careful to specifically delineate which use cases were attempting to support, and also not try to assume one implies the other. In other words, DPUs do offer some interesting capabilities towards Composible Hardware, however it is inherently not full configuration as the underlying host is still a static entity.

Note

DPUs are also sometimes expressed as xPUs, because classical graphics cards are Graphics Processing Units. While there does not appear to be any explicit movement into supporting that specific offload, some vendors are working on highly specific processing cards such those as performing protocol/signal translation. They may, or may not be able to have an operating system or provisioned application.

The problem

Today, In Ironic, we have a concept of a baremetal node. Inside of that baremetal node, there may be various hardware which can be centrally managed, and interacted with. It has a single BMC which controls basically all aspects.

We also have a concept of a smart nic in the form of `is_smartnic` on port objects. However this only impacts Ironics power and boot mode management.

Except, not all of these cards are network cards, or at least network cards in any traditional computing model. Think Radios!

And the intertwined challenge is this nested access model.

For the purpose of this example, Im going to refer to Nvidia Bluefield2 cards with a BMC. It should be noted we have support in Antelope IPA to update the BMC firmware of these cards. At least, that is our understanding of the feature.

But to do so:

1) From the host, the access restrictions need to be dropped by requesting the BMC on the card to permit the overall host OS to Access the cards BMC. This is achievable with an IPMI raw command, but against the cards BMC.

2) Then you would apply BMC firmware updates, to the cards BMC Today this would boot IPA, and perform it from the host OS, which also means that were going to need to interact with the overall host BMC, and boot the baremetal machine overall.

3) Then you would naturally want to drop these rights, which requires calling out to the cards BMC to change the access permissions.

4) Then if you wanted to update the OS on the card itself, you would rinse and repeat the process, with a different set of commands to open the access between the OS on the card, and the OS on the BMC.

Note

We know the Bluefield2 cards can both be network booted, and updated by SSHing into the BMC and streaming the new OS image to the installer command over SSH. That, itself, would be a separate RFE or feature but overall modeling changes would still be needed which this specification seeks to resolve.

Hopefully this illustrates the complexity, begins to underscore the need as to why we need to begin to support a parent/child device model, and permit the articulation of steps upon a parent node which applies to the one or more children nodes.

What complicates this further is is ultimately were just dealing with many different Linux systems, which have different models of access. For example, A more recent Dell Server running IPA, with two of these specalized cards, henceforth referred to as Data Processing Units (DPUs), would have Linux running on the host BMC, On the host processors, inside the BMCs of each DPU card, and inside of the processor on the DPU card.

This specification does inherently exclude configuration of the operating state configuration of the DPU card. There are other projects which are working on that, and we look forward to integrating with them as they evolve.

Note

The other project in mind is the OPI project, which is working on quite a lot of capabilities in this space, however they explicitly call out automated/manual deployment via outsize of zero touch provisioning is out of scope for their project. Such is sensible to stand-up a general purpose workload, but operating lifecycle and on-going management is an aspect where Ironic can help both operators who run a variety of workloads and configurations, or need to perform more specific lifecycle operations.

Proposed change

The overall idea with this specification is to introduce the building blocks to enable the orchusrtration and articulation of actions between parent and child devices.

- Introduction of `parent_node` field on the node object with an API version increase.
- Introduction of a sub-node resource view of `/v1/nodes/<node>/children` which allows the enumeration of child nodes.
- Default the /v1/nodes list to only list nodes without a parent, and add a query filter to return nodes with parents as well.
- Introduction of a new step field value, `execute_on_child_nodes` which can be submitted. The default value is False. Steps which return CLEANWAIT, i.e. steps which expect asynchronous return will not be permitted under normal conditions, however this will be available via a configuration option.
- Introduction of a new step field value, `limit_child_node_execution`, which accepts a list of node UUIDs to allow filtering and constraint of steps on some nodes. Specifically, this is largely separate from the `execute_on_child_nodes` field due to JSON Schema restrictions.
- Introduction of the ability to call a vendor passthrough interface as a step. In the case of some smartnics, they need the ability to call IPMI raw commands across child nodes.

- Introduction of the ability to call `set_boot_device` as a step. In this case, we may want to set the DPU cards to PXE boot en-mass to allow for software deployment in an IPA ramdisk, or other mechanism.
- Introduction of the ability to call `power_on`, `power_off` management interface methods through the conductor set_power_state helpers (which includes guarding logic for aspects like fast track).
- Possibly: Consider physical network interfaces optional for some classes of nodes. We wont know this until we are into the process of implementation of the capabilities.
- Possibly: Consider the machine UUID reported by the BMC as an identifier to match for agent operations. This has long been passively desired inside of the Ironic community as a nice to have.
- Possibly: We *may* need to continue to represent a parent before child or child before parent power management modeling like we did with the Port object `is_smartnic` field. This is relatively minor, and like other possible changes, we wont have a good idea of this until we are further along or some community partners are able to provide specific feedback based upon their experiences.

With these high level and workflow changes, it will be much easier for an operator to orchestrate management actions across an single node to extend further into distinct devices within the whole of the system.

In this model, the same basic rules for child nodes would apply, they may have their own power supplies and their own power control, and thus have inherent on and off states, so deletion of a parent should cause all child nodes to be deleted. For the purpose of state tracking, the individual cards if managed with a specific OS via Ironic, may be moved into a deployed state, however they may just also forever be in a `manageable` state independent of the parent node. This is because of the overall embedded nature, and it being less of less of a general purpose compute resource compute resource while *still* being a general computing device. This also sort of reflects the inherent model of it being more like firmware management to update these devices.

Outstanding Questions

- Do we deprecate the port object field `is_smartnic`? This is documented as a field to be used in the wild for changing the power/boot configuration flow on first generation smartnics which is still applicable on newer generations of cards should the operator have something like Neutron OVS agent connected on the message bus to allow termination of VXLAN connections to the underlying hardware within the card.

Out of Scope, for this Specification

Ideally, we do eventually want to have DPU specific hardware types, but the idea of this specification is to build the substrate needed to build upon to enable DPU specific hardware types and enable advanced infrastructure operators to do the needful.

Alternatives

Three alternatives exist. Technically four.

Do nothing

The first option is to do nothing, and force administrators to manage their nested hardware in a piecemeal fashion. This will create a barrier to Ironic usage, and we already know from some hardware vendors who are utilizing these cards along side Ironic, that the existing friction is a problem point in relation to

just power management. Which really means, this is not a viable option for Ironics use in more complex environments.

Limit scope and articulate specific workflows

A second option is to potentially limit the scope of support to just power or boot operations. However, we have had similar discussions, in relation to supporting xPUs in servers with external power supplies in the past, and have largely been unable to navigate a workable model, in large part because this model would generally require a single task.node to be able to execute with levels of interfaces with specific parameters. For example, to the system BMC for base line power management, and then to a SNMP PDU for the auxiliary power. This model also doesnt necessarily work because then we would inherently have blocked ourselves from more general managmeent capabilities and access to on DPU card features such as serial consoles through its own embedded BMC without substantial refactoring and re-doing the data model.

There is also the possibility that nesting access controls/modeling may not be appropriate. You dont necessarily want to offer an baremetal tenant in a BMaaS who has lessee access to Ironic, the ability to get to a serial console which kind of points us to the proposed solution in order to provide capabilities to handle the inherently complex nature of modeling which can result. Or eat least provide organic capabilities based upon existing code.

Use Chassis

The third possibility is to use the existing Chassis resource. The idea of a parent/child relationship *does* sound similar to the modeling of Chasssis and a Node.

Chassis was originally intended to allow the articulation of entire Racks or Blade Chassis in Ironics data model, in part to allow relationship and resource tracking more in lines with a Configuration Management Data Base (CMDB) or Asset Inventory. However, Chassis never gained much traction because those systems are often required and decoupled in enterprise environments.

Chassis has been proposed to be removed several times in Ironic, and does allow the creation of a one to many relationship which cannot presently be updated after it is set. Which inherently is problematic and creates a maintenanance burden should a card need to be moved or a chassis replaced but the DPU is just moved to the new chassis.

But the inherent one to many modeling which can exist with DPUs ultimately means that the modeling is in reverse from what is implemented for usage. Nodes would need to be Chassises, but then how do users schedule/deploy instances, much less perform targeted lifecycle operations against part of the machine which is independent, and can be moved to another chassis.

Overall, this could result in an area where we may make less progress because we would essentially need to re-model the entire API, which might be an interesting challenge, but that ultimately means the work required is substantially larger, and we would potentially be attempting to remodel interactions and change the user experience, which means the new model would also be harder to adopt with inherently more risk if we do not attempt to carry the entire feature set to a DPU as well. If we look at solving the overall problem from a reuse standpoint, the proposed of this specification document solution seems like a lighter weight solution which also leaves the door open to leverage the existing capabilities and provide a solid foundation for future capabilities.

Realistically, The ideal use case for chassiss is fully composible hardware where some sort of periodic works to pre-populate available nodes to be scheduled upon by services like Nova from a pool of physical resources, as well as works to reconcile overall differences. The blocker though to that is ultimately

availability of the hardware and related documentation to make a realistic Chassis driver happen in Open Source.

Create a new interface or hardware type

We could create a new interface on a node, or a new hardware type.

We do eventually want some DPU specific items to better facilitate and enable end operators, however there is an underling issue of multiple devices, a one to many relationship. Further complicated by a single machine may have a number of different types of cards or devices, which kind of points us back to the original idea proposed.

Data model impact

A `parent_node` field will be added, and the field will be indexed. A possibility exists that the DB index added may be a multi-field compound index as well, but that is considered an implementation detail.

State Machine Impact

No State Machine impact is expected.

REST API impact

GET /v1/nodes/?include_children=True

Returns a list of base nodes with all child nodes child nodes, useful for a big picture view of all things Ironic is responsible for.

GET /v1/nodes/

The view will by default return only nodes where the `parent_node` field is null. Older API clients will still receive this default behavior change.

GET /v1/nodes/<node_ident>/children

Will return the list of nodes, with the pre-existing access list constraints and modeling of all defined nodes where `parent_node` matches `node_ident`. In alignment with existing node list behavior, if access rights do not allow the nodes to be viewed, or there are no nodes, an empty list will be returned to the API client.

Additional parameters may also be appropriate with this field, but at present they are best left to be implementation details leaning towards the need to not support additional parameters.

Note

We would likely need to validate the submitted node_ident is also a UUID, otherwise resolve the name to a node, and then lookup the UUID.

A links field will refer to each node, back to the underlying node which may require some minor tuning of the logic behind node listing and link generation.

All of the noted changes should be expected to be merged together with a microversion increase. The only non-version controlled change, being the presence/match of the `parent_node` field.

Corresponding API client changes will be needed to interact with this area of the code.

Client (CLI) impact

openstack baremetal CLI

The `baremetal` command line interface will need to receive parameters to query child nodes, and query the child nodes of a specific node.

openstacksdk

An SDK change may not be needed, or may be better suited to occur organically as someone identifies a case where they need cross-service support.

RPC API impact

No additional RPC API calls are anticipated.

Driver API impact

No direct driver API changes are anticipated as part of this aside from ensureing the management interface `set_boot_device` as well as the IPMI interface `send_raw` commands can be called via the steps framework.

Nova driver impact

None are anticipated, this is intended to be invisible to Nova.

Ramdisk impact

The execution of our ramdisk inside of a DPU is presently considered out of scope at this time.

Some of the existing smartnics might not be advisable to have operations like cleaning as well, for example Bluefield2 cards with more traditional SPI flash as opposed to NVMe in Bluefield3 cards. Given some of the speciality in methods of interacting with such hardware, we anticipate we may eventually want to offer specific deployment or boot interfaces which may bypass some of the inherent agent capabilities.

Security impact

No additional security impact is anticipated as part of this change.

Other end user impact

None

Scalability impact

This change does propose an overall relationship and ability which may result far more nodes to be managed in ironics database. It may also be that for child devices, a power synchronization loop may *not* be needed, or can be far less frequent. These are ultimately items we need to discuss further, and consider some additional controls if we determine the need so operators may not feel any need nor impact to their deployments due to the increase in rows int the nodes table.

Note

It should be noted that the way the hash ring works in Ironic, is that the ring consists of the *conductors*, which are then mapped to based upon node properties. It may be that a child nodes mapping should be the parent node. These are questions to be determined.

Performance Impact

No direct negative impact is anticipated. The most direct impact will be the database and some periodics which we have already covered in the preceding section. Some overall performance may be avoided by also updating some of the periodics to not possibly match any child node, the logical case is going to be things like RAID periodics, which would just never apply and should be never configured for such a device, which may itself make the need to make such a periodic change moot.

Other deployer impact

No negative impact is anticipated, but it might be that operators may rapidly identify need for a BMC SSH Command interface, as the increasing trend of BMCs being linux powered offers increased capabilities and possibilities, along with potential needs if logical mappings do not map out.

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
Julia (TheJulia) Kreger <juliaashleykreger@gmail.com>

Other contributors:
<IRC handle, email address, None>

Work Items

- Addition of `parent_node` db field and node object.
- Addition of node query functionality.
- Introduction of the /v1/nodes/<node>/children API resource and the resulting API microversion increase.
- Add step support to iterate through step definitions which has mixed step commands for parent nodes and child node.
- Introduction of generalized power interface steps: * `power_on` * `power_off`
- Add an IPMI management interface `raw` command step method.
- Examples added for new step commands and invocation of child node objects.

Dependencies

None.

Testing

Basic tempest API contract testing is expected, however a full tempest scenario test is not expected.

Upgrades and Backwards Compatibility

No negative impact is anticipated.

Documentation Impact

Documentation and examples are expected as part of the work items.

References

- https://github.com/opiproject/opi-prov-life/blob/main/PROVISIONING.md#additional-provisioning-methods-out-of-opi-scope
- https://docs.nvidia.com/networking/display/BlueFieldBMCSWLatest/NVIDIA+OEM+Commands

5.15.196 Support for Software RAID

https://storyboard.openstack.org/#!/story/2004581

This spec proposes to add support for the configuration of software RAIDs.

In analogy to the way hardware RAIDs are currently set up, the RAID setup shall be done as part of the cleaning (clean-time software RAID). Admin Users define the target RAID config which will be applied whenever the node is cleaned, i.e. before it becomes available for instance creation.

In order to allow the End User to provide details on how the software RAID shall be configured, the RAID setup should eventually become part of the deployment steps. Integrating this into the deployment steps framework, however, is beyond the scope of this spec.

Problem description

As it is hardware agnostic, flexible, reliable, and easy to use, software RAID has become a popular choice to protect against disk device failures - also in production setups. Large deployments, such as the ones at Oath or CERN, rely on software RAID for their various services.

Ironics current lack of support for such setups requires Deployers and Admins to withdraw to workarounds in order to provide their End Users with physical instances based on a software RAID configuration. These workarounds may require to maintain an additional installation infrastructure which is then either integrated into the installation process or requires the End User to re-install a machine a second time after it has been already provisioned by Ironic to eventually end up with the desired configuration of the disk devices. This increases the complexity for Deployers and Admins, and can also lead to a decrease of the End Users satisfaction with the overall provisioning and installation process.

Proposed change

The proposal is to extend Ironic to support software RAID by:

- using a nodes `target_raid_config` to specify the desired s/w RAID layout (with some restrictions, see below);
- adding support in the `ironic-python-agent` to understand a software RAID config as specified in a nodes `target_raid_config` and be able to create and delete such configurations;
- allow the `ironic-python-agent` to consider s/w RAID devices for deployment, e.g. via root device hints (considering them at all is already addressed in [1]);
- adding support in Ironic and the `ironic-python-agent` to take the necessary steps to boot from a s/w RAID, e.g. installing the boot loader on the correct device(s).

Initially, only the following configurations will be supported for the `target_raid_config` as to be set by the Admin:

- a single RAID-1 spanning the available devices and serving as the deploy target device, or
- a RAID-1 serving as the deploy target device plus a RAID-N where the RAID level N is configurable by the Admin. N can be 0, 1, 5, 6, or 10.

The supported configurations have been limited to these two options in order to avoid issues when booting from RAID devices. Having a (small) RAID-1 device to boot from is a common approach when setting up more advanced RAID configurations: a RAID-1 holder device can look like a standalone disk and does not require the bootloader to have any knowledge or capabilities to understand more complex RAID configurations.

In order to signal that a software RAID configuration is indeed desired (and to protect from a situation where a software RAID is set up accidentally when the configuration passed via the `target_raid_config` was meant for a hardware RAID setup, for instance), the `controller` property of all of the logical disks needs to be set to `software`. Without this setting, the software RAID code in the GenericHardwareManager of the IPA will ignore the given `target_raid_config`. If it is set on only one of the logical drives, the validation code will raise an error.

The `controller` property set to `software` will also be used by the conductor to identify a software RAID and trigger the required installation of the bootloader. While whole-disk images are expected to come with a bootloader configuration as part of the image, for software RAIDs in the current design the image will not be at the start of a real disk, but inside the first partition on top of a software RAID-1. The bootloader must hence be explicitly installed onto the underlying holder disks, and this property will indicate when to do this.

An example of a valid software RAID configuration would hence look like:

```
{
    "logical_disks": [
        {
            "size_gb": 100,
            "raid_level": "1",
            "controller": "software"
        },
        {
            "size_gb": "MAX",
            "raid_level": "0",
            "controller": "software"
```

(continues on next page)

(continued from previous page)

```
        }
    ]
}
```

Support for more than one RAID-N, support for the selection of a subset of drives to act as holder devices, support for simultaneous software and hardware RAID devices as well as support to partition the created RAID-N device are left for follow-up enhancements and beyond the scope of this specification.

Also, there is currently no support for partition images, only whole disk images are supported.

A first prototype very close to the proposal is available from [2][3][4].

Alternatives

As mentioned above, the alternative is to use other methods to create s/w RAID setups on physical nodes and integrate these out-of-band approaches into the provisioning workflow of individual deployments. This increases complexity on the Deployer/Admin side and can have a negative impact on the user experience when creating physical instances which need to have a software RAID setup..

Data model impact

None.

State Machine Impact

None.

REST API impact

None.

Client (CLI) impact

None.

ironic CLI

None.

openstack baremetal CLI

None.

RPC API impact

None.

Driver API impact

The proposed functionality could be consolidated into a new RAID interface.

Nova driver impact

None.

Ramdisk impact

The `ironic-python-agent` will need to be able to: * setup and clean software RAID devices * consider software RAID devices for deployment * configure the holder devices of the RAID-1 device in a way they are bootable

This functionality could be consolidated in an additional RAID interface.

Security impact

None.

Other end user impact

While the predefined RAID-1 ensures that a system should be able to boot, End Users need to be aware that the kernel of the started image needs to be able to understand software RAID devices.

Scalability impact

None.

Performance Impact

None.

Other deployer impact

Deployers will need to be aware that the configuration and clean up of the RAID-N devices is only done during cleaning, so any changes require the node to be cleaned. Also, the config is not configurable by the End User, but limited to admins (as the target_raid_config) is a node property. All of this, however, already holds true for hardware RAID configurations.

Developer impact

None.

Implementation

An initial proof-of-concept is available from [2][3][4].

Assignee(s)

Primary assignee:
None.

Other contributors:
Arne.Wiebalck@cern.ch (arne_wiebalck)

Work Items

This is to be defined once the overall idea is accepted and theres agreement on a design.

Dependencies

None.

Testing

TBD

Upgrades and Backwards Compatibility

None.

Documentation Impact

Documentation on how to configure a software RAID along with the limitations outlined in Deployers Impact need to be documented.

References

[1] https://review.opendev.org/#/c/592639 [2] CERN Hardware Manager: https://github.com/cernops/cern-ironic-hardware-manager/commit/7f6d892ec4848a09000ed1f28f3137bf8ba917f0 [3] Patched Ironic Python Agent: https://github.com/cernops/ironic-python-agent/commit/bddac76c4d100af0103a6bc08b81dd71681a9c02 [4] Patched Ironic: https://github.com/cernops/ironic/commit/581e65f1d8986ac3e859678cb9aadd5a5b06ba60

5.15.197 SSH Console

https://bugs.launchpad.net/ironic/+bug/1526305

This implements console driver ShellinaboxConsole which supports console access for SSH driver (only for virsh virt_type).

Problem description

Currently there is no support to get the console for virtual machines in dev and test environments.

Proposed change

Implements a console driver ShellinaboxConsole that uses ssh+shellinabox to connect to console of virtual machines:

- Use existing ironic/drivers/modules/console_utils module to start/stop shellinabox.
- Add ssh_terminal_port property to CONSOLE_PROPERTIES. This is going to be port on which shellinabox listens locally.
- Add new class ShellinaboxConsole inherited from base.ConsoleInterface in ironic/drivers/modules/ssh.py
- **Implement the following methods in ShellinaboxConsole class.**
  - **`validate()` - Validate the Node console info.**
    - param task : a task from TaskManager.

        - raises : InvalidParameterValue.
        - raises : MissingParameterValue.
    - `start_console()` - Start a remote console for the node.
    - `stop_console()` - Stop the remote console session for the node.
    - `get_console()` - Get the type and connection information about the console.
- Add self variable self.console in PXEAndSSHDriver and AgentAndSSHDriver.

Alternatives

None

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

N/A

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

The following which is already part of driver_info field is required:

- `ssh_address` - IP address or hostname of the node to ssh into.
- `ssh_username` - Username to authenticate as.
- `ssh_virt_type` - Virtualization software to use; must be virsh.

Additionally one field need to be provided with driver_info

- `ssh_terminal_port` - Port to connect to, only required for console access.

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
niu-zglinux

Work Items

Implement `ShellinaboxConsole` class inherited from `base.ManagementInterface`. Implement `validate` , `start_console`, `stop_console`, `get_console`. Add ability to enable pty console in devstack scripts, and leave the log console by default in order not to affect the gate logs.

Dependencies

None

Testing

Unit Testing will be added.

Upgrades and Backwards Compatibility

None

Documentation Impact

None

References

None

5.15.198 Add Intelő Speed Select Support

https://storyboard.openstack.org/#!/story/2005390

Multiple types of servers are needed to handle diverse workloads. Purchasing and managing these servers introduces complexity and increases total cost of ownership(TCO). Intel Speed Select Technology(SST)[1] is a collection of features that improves performance and optimizes TCO by providing more control over CPU performance. With Intel SST, one server can do more which means the same server can be used to run different workloads. One of the feature is Intel Speed Select Technology-Performance Profile (SST-PP) which allows configuring the CPU to run at 3 distinct operating points or profiles.

With Intel SST-PP, one can set the desired core count and their base and turbo frequencies which can help to tune the server to specific workload performance.

Problem description

This spec proposes to support Intel SST-PP feature in Ironic. With Intel SST-PP, Ironic users can:

- Run their servers at different configuration level.
- Each configuration level supports different number of active cores and frequency.
- Same server can be used to run multiple different workloads thus decreasing TCO.

Intel SST-PP Speed Select supports three configuration levels:

- 0 - Intel SST-PP Base Config
- 1 - Intel SST-PP Config 1
- 2 - Intel SST-PP Config 2

Following table shows the list of active cores and their base frequency at different SST-PP config levels:

```
-------------------------------------------
|   Config    |  Cores  |  Base Freq (GHz)  |
-------------------------------------------
| Base        | 24      | 2.4               |
| Config 1    | 20      | 2.5               |
| Config 2    | 16      | 2.7               |
-------------------------------------------
```

Proposed change

Intel SST-PP can be set over IPMI. Each configuration level has its own hexa raw code that the server understands. Ironic sends this code to the server via IPMI to set the desired SST-PP level.

We will map these configurations to traits that Ironic understands.

- 0 - `CUSTOM_INTEL_SPEED_SELECT_CONFIG_BASE`
- 1 - `CUSTOM_INTEL_SPEED_SELECT_CONFIG_1`
- 2 - `CUSTOM_INTEL_SPEED_SELECT_CONFIG_2`

The solution works at two stages:

Scheduling

The solution to support Intel SST-PP is first enable scheduling the request for baremetal instance deployment on the nodes that supports it. We set the desired configuration as the trait in our flavor:

```
$ openstack flavor set --property \
  trait:CUSTOM_INTEL_SPEED_SELECT_CONFIG_2=required baremetal
```

Now, we also need to update the Ironic nodes trait with the supported configuration levels:

```
$ openstack baremetal node add trait node-0 \
  CUSTOM_INTEL_SPEED_SELECT_CONFIG_BASE CUSTOM_INTEL_SPEED_SELECT_CONFIG_1 \
  CUSTOM_INTEL_SPEED_SELECT_CONFIG_2
```

Now, when user sends a request to boot a node with the `baremetal` flavor, placement API service will select the Ironic node that supports Intel SST-PP.

Provisioning

The Intel SST-PP needs to be set via IPMI before powering on the node in the process of provisioning. Ironic API service receives the desired configuration in `node.instance_info.traits` in the boot request. Ironic will then run the deploy templates step matching the trait. The deploy template will specify the new `configure_intel_speedselect` step which configures the Intel SST-PP configuration level and then powers on the node.

Ironic will need below details to configure Intel SST-PP on the node:

- Intel SST-PP configuration level: This is the required configuration level on the node. Possible values are the traits listed above. This information is set by admins in the flavors trait they want to boot the baremetal node with. Nova in turn updates the Ironics node information with the trait.
- Number of sockets: This is the number of sockets per CPU. Setting Intel SST-PP needs to be done for every socket.

Both these information can be provided as an argument to the `configure_intel_speedselect` deploy step.

Alternatives

Continue to not support Intel SST-PP or users can manually configure it independent of Ironic.

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

ironic CLI

None

openstack baremetal CLI

None

RPC API impact

None

Driver API impact

- Add a new hardware type `IntelIPMIHardware` to support Intel SST-PP enabled servers.

```
class IntelIPMIHardware(IPMIHardware):
```

- Add a new management interface `IntelIPMIManagement` to manage the configuration of Intel SST-PP on the servers. This class will have a new deploy step `configure_intel_speedselect` to configure Intel SST-PP on the nodes. This will also be enabled as a clean step so that it can be used to reset the configuration while node cleaning.

```
class IntelIPMIManagement(ipmitool.IPMIManagement):
    @base.deploy_step(priority=200, argsinfo={
        'intel_speedselect_config': {
            'description': (
                 "Intel SST-PP configuration."
            ),
            'required': True
        },
        'socket_count': {
            'description': (
                "No. of sockets."
```

(continues on next page)

(continued from previous page)

```
            )
        }
    })
    def configure_intel_speedselect(self, task, **kwargs):
        return None
```

The following table contains the `intel_speedselect_config` values for the proposed deploy templates:

Deploy Template	intel_speedselect_config
CUSTOM_INTEL_SPEED_SELECT_CONFIG_BASE	0x00
CUSTOM_INTEL_SPEED_SELECT_CONFIG_1	0x01
CUSTOM_INTEL_SPEED_SELECT_CONFIG_2	0x02

Nova driver impact

None

Ramdisk impact

None

Security impact

None

Other end user impact

Users will have to update the traits with the desired Intel SST-PP configuration level in the corresponding flavors.

Scalability impact

None

Performance Impact

None

Other deployer impact

Deployers wishing to use feature will have to add the Intel SST-PP configuration in Nodes trait and also have to create corresponding deploy templates.

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
mkrai

Work Items

- Implement Intel SST-PP support code in Ironic.
- Write the test code.
- Write a document explaining how to use Intel SST-PP.

Dependencies

None

Testing

- Unit tests

Upgrades and Backwards Compatibility

None

Documentation Impact

New document will be added to explain the support of new Intel SST-PP feature in Ironic and how to use it.

References

[1] https://www.intel.com/content/www/us/en/architecture-and-technology/speed-select-technology-article.html

5.15.199 Support Rolling Upgrade

https://bugs.launchpad.net/ironic/+bug/1526283

This proposes support for rolling upgrades of ironic, which will allow operators to roll out new code to the ironic services without having to restart all services on the new code simultaneously. There could be minimal downtime when upgrading ironic services. During the upgrade of ironic, instances will not be impacted; they should continue to run and have network access. There might be slightly longer delays for ironic operations to be performed on baremetal nodes though.

This support for rolling upgrade will satisfy the criteria mentioned in the OpenStack governance[1].

[1] https://github.com/openstack/governance/blob/master/reference/tags/assert_supports-rolling-upgrade.rst

Problem description

People operating an OpenStack cloud face the same problem: how to upgrade the cloud to the latest version so as to enjoy the new features and changes.

The upgrade of a cloud is typically done one component at a time. For example, these components of OpenStack could be upgraded in this order:

1. upgrade keystone
2. upgrade glance
3. upgrade ironic
4. upgrade nova
5. upgrade neutron
6. upgrade cinder

Ironic already supports cold upgrades[14], where the ironic services have to be down during the upgrade. For time-consuming upgrades, it may be unacceptable for the services to be unavailable for a long period of time. The steps[13] to do a cold upgrade could be:

1. stop all ironic-api and ironic-conductor services
2. uninstall old code
3. install new code
4. update configurations
5. DB sync (most time-consuming task)
6. start ironic-api and ironic-conductor services

A rolling upgrade would provide a better experience for the users and operators of the cloud. In the context of ironics rolling upgrade, it means it wouldnt be necessary to upgrade all the ironic-api and ironic-conductor services simultaneously as in a cold upgrade. A rolling upgrade would allow individual ironic-api and ironic-conductor services to be upgraded one at a time, with the rest of the services still available. This upgrade would have minimal downtime.

Although wed like the rolling upgrade solution presented here to be the same as that used by novas upgrade process[2], there are some differences between nova and ironic that prevent us from using the same solution. The differences are mentioned below, in other sections of this specification.

There have been several discussions about rolling upgrades ([3],[9],[10]).

Proposed change

For a rolling upgrade to have minimal downtime, there should be at least two ironic-api services and two ironic-conductor services running.

Since ironic doesnt support database downgrades, rollbacks will not be supported.

To support rolling upgrade for ironic, the following are needed:

[14] https://github.com/openstack/governance/blob/master/reference/tags/assert_supports-upgrade.rst
[13] http://docs.openstack.org/developer/ironic/deploy/upgrade-guide.html
[2] http://docs.openstack.org/developer/nova/upgrade.html
[3] https://etherpad.openstack.org/p/ironic-mitaka-midcycle
[9] http://lists.openstack.org/pipermail/openstack-dev/2016-April/092773.html
[10] https://etherpad.openstack.org/p/ironic-newton-summit-live-upgrades

- code changes
- a contribution guide for developers and reviewers
- a multi-node grenade CI to prevent breaking the mechanism of rolling upgrade
- a rolling upgrade operation guide for operators.

The following sub-sections describe:

- support for rolling upgrades between ironic releases
- the proposed rolling upgrade process
- the changes that are needed

Rolling upgrades between releases

Ironic follows the release-cycle-with-intermediary release model[6]. The releases are semantic-versioned[7], in the form <major>.<minor>.<patch>. We refer to a named release of ironic as the release associated with a development cycle like Mitaka.

In addition, ironic follows the standard deprecation policy[8], which says that the deprecation period must be at least three months and a cycle boundary. This means that there will never be anything that is both deprecated *and* removed between two named releases.

Rolling upgrades will be supported between:

- Named release N to N+1. (N would start with Newton if this feature is merged in Ocata.)
- Any named release to its latest revision, containing backported bug fixes. Because those bug fixes can contain improvements to the upgrade process, the operator should patch the system before upgrading between named releases.
- Most recent named release N (and semver releases newer than N) to master. As with the above bullet point, there may be a bug or a feature introduced on a master branch, that we want to remove before publishing a named release. Deprecation policy allows to do this in a 3 month time frame[8]. If the feature was included and removed in intermediate releases, there should be a release note added, with instructions on how to do a rolling upgrade to master from an affected release or release span. This would typically instruct the operator to upgrade to a particular intermediate release, before upgrading to master.

Rolling upgrade process

The rolling upgrade process to upgrade ironic from version `FromVer` to the next version `ToVer` is as follows:

1. Upgrade Ironic Python Agent image before upgrading ironic.
2. Upgrade DB schema to `ToVer` via **ironic-dbsync upgrade**. Ironic already has the code in place to do this. However, a new DB migration policy (described below in *New DB model change policy*) needs to be documented.
3. Pin RPC and IronicObject versions to the same `FromVer` for both ironic-api and ironic-conductor services, via the new configuration option described below in *RPC and object version pinning*.

[6] https://releases.openstack.org/reference/release_models.html
[7] http://semver.org/
[8] http://governance.openstack.org/reference/tags/assert_follows-standard-deprecation.html

4. Upgrade code and restart ironic-conductor services, one at a time.
5. Upgrade code and restart ironic-api services, one at a time.
6. Unpin RPC and object versions so that the services can now use the latest versions in `ToVer`. This is done via updating the new configuration option described below in *RPC and object version pinning* and then restarting the services. ironic-conductor services should be restarted first, followed by the ironic-api services. This is to ensure that when new functionality is exposed on the unpinned API service (via API micro version), it is available on the backend.
7. Run a new command **ironic-dbsync online_data_migration** to ensure that all DB records are upgraded to the new data version. This new command is discussed in a separate RFE[12] (and is a dependency for this work).
8. Upgrade ironic client libraries (e.g. python-ironicclient) and other services which use the newly introduced API features and depend on the new version.

The above process will cause the ironic services to be running the `FromVer` and `ToVer` releases in this order (where step refers to the steps above):

step	ironic-api	ironic-conductor
0	all FromVer	all FromVer
4.1	all FromVer	some FromVer, some ToVer-pinned
4.2	all FromVer	all ToVer-pinned
5.1	some FromVer, some ToVer-pinned	all ToVer-pinned
5.2	all ToVer-pinned	all ToVer-pinned
6.1	all ToVer-pinned	some ToVer-pinned, some ToVer
6.2	all ToVer-pinned	all ToVer
6.3	some ToVer-pinned, some ToVer	all ToVer
6.4	all ToVer	all ToVer

New DB model change policy

This is not a code change but it impacts the SQLAlchemy DB model and needs to be documented well for developers as well as reviewers. This new DB model change policy is as follows:

- Adding new items to the DB model is supported.
- The dropping of columns/tables and corresponding objects fields is subject to ironic deprecation policy[Page 1448, 8]. But its alembic script has to wait one more deprecation period, otherwise an unknown column exception will be thrown when `FromVer` services access the DB. This is because **ironic-dbsync upgrade** upgrades the DB schema but `FromVer` services still contain the dropped field in their SQLAlchemy DB model.
- alter_column like rename or resize is not supported anymore. This has to be split into multiple operations, like add column, then remove column. Some changes may have to be split into multiple releases to maintain compatibility with an old SQLAlchemy model.
- some implementations of ALTER TABLE like adding foreign keys in PostgreSQL may impose table locks and cause downtime. If the change cannot be avoided and the impact is significant (the table can be frequently accessed and/or store a large dataset), these cases must be mentioned in the release notes.

[12] http://bugs.launchpad.net/ironic/+bug/1585141

RPC and object version pinning

For the ironic (ironic-api and ironic-conductor) services to be running old and new releases at the same time during a rolling upgrade, the services need to be able to handle different RPC versions and object versions.

[4] has a good description of why we need RPC versioning, and describes how nova deals with it. This proposes taking a similar approach in ironic.

For object versioning, ironic uses oslo.versionedobjects.[5] describes novas approach to the problem. Unfortunately, ironics solution is different, since ironic has a more complex situation. In nova, all database access (reads and writes) is done via the nova-conductor service. This makes it possible for the nova-conductor service to be the only service to handle conversions between different object versions. (See[5] for more details.) Given an object that it doesnt understand, a (non nova-conductor) service will issue an RPC request to the nova-conductor service to get the object converted to its desired target version. Furthermore, for a nova rolling upgrade, all the non-nova-compute services are shut down, and then restarted with the new releases; nova-conductor being the first service to be restarted (Page 1447, 2). Thus, the nova-conductor services are always running the same release and dont have to deal with differing object versions amongst themselves. Once they are running the new release, they can handle requests from other services running old or new releases.

Contrast that to ironic, where both the ironic-api and ironic-conductor services access the database for reading and writing. Both these services need to be aware of different object versions. For example, ironic-api can create objects such as Chassis, Ports, and Portgroups, saving them directly to the database without going through the conductor. We cannot take down the ironic-conductor in a similar way as the nova-conductor service, because ironic-conductor does a whole lot more than just interacting with the database, and at least one ironic-conductor needs to be running during a rolling upgrade.

A new configuration option will be added. It will be used to pin the RPC and IronicObject (e.g., Node, Conductor, Chassis, Port, and Portgroup) versions for all the ironic services. With this configuration option, a service will be able to properly handle the communication between different versions of services.

The new configuration option is: `[DEFAULT]/pin_release_version`. The default value of empty indicates that ironic-api and ironic-conductor will use the latest versions of RPC and IronicObjects. Its possible values are releases, named (e.g. `ocata`) or sem-versioned (e.g. `7.0`).

Internally, ironic will maintain a mapping that indicates the RPC and IronicObject versions associated with each release. This mapping will be maintained manually. (It is possible, but outside the scope of this specification, to add an automated process for the mapping.) Here is an example:

- objects_mapping:

```
{'mitaka': {'Node': '1.14', 'Conductor': '1.1',
            'Chassis': '1.3', 'Port': '1.5', 'Portgroup': '1.0'},
 '5.23': {'Node': '1.15', 'Conductor': '1.1',
          'Chassis': '1.3', 'Port': '1.5', 'Portgroup': '1.0'}}
```

- rpc_mapping:

```
{'mitaka': '1.33', '5.23': '1.33'}
```

During a rolling upgrade, the services using the new release should set this value to be the name (or version) of the old release. This will indicate to the services running the new release, which RPC and

[4] http://superuser.openstack.org/articles/upgrades-in-openstack-nova-remote-procedure-call-apis
[5] http://superuser.openstack.org/articles/upgrades-in-openstack-nova-objects/

object versions that they should be compatible with, in order to communicate with the services using the old release.

Handling RPC versions

`ConductorAPI.__init__()` already sets a `version_cap` variable to the latest RPC API version and passes it to the `RPCClient` as an initialization parameter. This `version_cap` is used to determine the maximum requested message version that the `RPCClient` can send.

In order to make a compatible RPC call for a previous release, the code will be changed so that the `version_cap` is set to a pinned version (corresponding to the previous release) rather than the latest `RPC_API_VERSION`. Then each RPC call will customize the request according to this `version_cap`.

Handling IronicObject versions

Internally, ironic services (ironic-api and ironic-conductor) will deal with IronicObjects in their latest versions. Only at these boundaries, when the IronicObject enters or leaves the service, will we need to deal with object versioning:

- *getting objects from the database*: convert to latest version
- *saving objects to the database*: if pinned, save in pinned version; else save in latest version
- *serializing objects (to send over RPC)*: if pinned, send pinned version; else send latest version
- *deserializing objects (receiving objects from RPC)*: convert to latest version

The ironic-api service also has to handle API requests/responses based on whether or how a feature is supported by the API version and object versions. For example, when the ironic-api service is pinned, it can only allow actions that are available to the objects pinned version, and cannot allow actions that are only available for the latest version of that object.

To support this:

- add a new column named `version` to all the database tables (SQLAlchemy models) of the IronicObjects. The value is the version of the object that is saved in the database.

  This version column will be null at first and will be filled with the appropriate versions by a data migration script. If there is a change in Ocata that requires migration of data, we will check for null in the new version column.

  No project uses the version column mechanism for this purpose, but it is more complicated without it. For example, Cinder has a migration policy which spans 4 releases in which data is duplicated for some time. Keystone uses triggers to maintain duplicated data in one release cycle. In addition, the version column may prove useful for zero-downtime upgrades (in the future).

- add a new method `IronicObject.get_target_version(self)`. This will return the target version. If pinned, the pinned version is returned. Otherwise, the latest version is returned.
- add a new method `IronicObject.convert_to_version(self, target_version)`. This method will convert the object into the target version. The target version may be a newer or older version that the existing version of the object. The bulk of the work will be done in the new helper method `IronicObject._convert_to_version(self, target_version)`. Subclasses that have new versions should redefine this to perform the actual conversions.
- add a new method `IronicObject.do_version_changes_for_db(self)`. This is described below in *Saving objects to the database (API/conductor > DB)*.

- add a new method `IronicObjectSerializer._process_object(self, context, objprim)`. This is described below in *Receiving objects via RPC (API/conductor <- RPC)*.

In the following,

- The old release is `FromVer`; it uses version 1.14 of a Node object.
- The new release is `ToVer`; it uses version 1.15 of a Node object this has a deprecated `extra` field and a new `meta` field that replaces `extra`.
- db_obj[meta] and db_obj[extra] are the database representations of those node fields.

Getting objects from the database (API/conductor < DB)

Both ironic-api and ironic-conductor services read values from the database. These values are converted to IronicObjects via the existing method `IronicObject._from_db_object(context, obj, db_object)`. This method will be changed so that the IronicObject will be in the latest version, even if it was in an older version in the database. This is done regardless of the service being pinned or not.

Note that if an object is converted to a later version, that IronicObject will retain any changes resulting from that conversion (in case the object later gets saved in the latest version).

For example, if the node in the database is in version 1.14 and has db_obj[extra] set:

- a `FromVer` service will get a Node with node.extra = db_obj[extra] (and no knowledge of node.meta since it doesnt exist).
- a `ToVer` service (pinned or unpinned), will get a Node with:
  - node.meta = db_obj[extra]
  - node.extra = None
  - node._changed_fields = [meta, extra]

Saving objects to the database (API/conductor > DB)

The version used for saving IronicObjects to the database is determined as follows:

- for an unpinned service, the object will be saved in its latest version. Since objects are always in their latest version, no conversions are needed.
- for a pinned service, the object will be saved in its pinned version. Since objects are always in their latest version, the object will need to be converted to the pinned version before being saved.

The new method `IronicObject.do_version_changes_for_db()` will handle this logic, returning a dictionary of changed fields and their new values (similar to the existing `oslo.versionedobjects.VersionedObjectobj.obj_get_changes()`). Since we do not keep track internally, of the database version of an object, the objects `version` field will always be part of these changes.

The *Rolling upgrade process* (at step 6.1) ensures that by the time an object can be saved in its latest version, all services are running the newer release (although some may still be pinned) and can handle the latest object versions.

An interesting situation can occur when the services are as described in step 6.1. It is possible for an IronicObject to be saved in a newer version and subsequently get saved in an older version. For example, a `ToVer` unpinned conductor might save a node in version 1.5. A subsequent request may cause a `ToVer` pinned conductor to replace and save the same node in version 1.4!

Sending objects via RPC (API/conductor -> RPC)

When a service makes an RPC request, any IronicObjects that are sent as part of that request are serialized into entities or primitives (via `oslo.versionedobjects.VersionedObjectSerializer.serialize_entity()`). The version used for objects being serialized is as follows:

- for an unpinned service, the object will be serialized in its latest version. Since objects are always in their latest version, no conversions are needed.
- for a pinned service, the object will be serialized in its pinned version. Since objects are always in their latest version, the object will need to be converted to the pinned version before being serialized. The converted object will include changes that resulted from the conversion; this is needed so that the service at the other end of the RPC request has the necessary information if that object will be saved to the database.

The `IronicObjectSerializer.serialize_entity()` method will be modified to do any IronicObject conversions.

Receiving objects via RPC (API/conductor <- RPC)

When a service receives an RPC request, any entities that are part of the request need to be deserialized (via `oslo.versionedobjects.VersionedObjectSerializer.deserialize_entity()`). For entities that represent IronicObjects, we want the deserialization process to result in IronicObjects that are in their latest version, regardless of the version they were sent in and regardless of whether the receiving service is pinned or not. Again, any objects that are converted will retain the changes that resulted from the conversion, useful if that object is later saved to the database.

The deserialization method invokes `VersionedObjectSerializer._process_object()` to deserialize and get the IronicObject. We will add `IronicObjectSerializer._process_object()` to convert the IronicObject to its latest version.

For example, a `FromVer` ironic-api could issue an update_node() RPC request with a node in version 1.4, where node.extra was changed (so node._changed_fields = [extra]). This node will be serialized in version 1.4. The receiving `ToVer` pinned ironic-conductor deserializes it and converts it to version 1.5. The resulting node will have node.meta set (to the changed value from node.extra in v1.4), node.extra = None, and node._changed_fields = [meta, extra].

Alternatives

A cold upgrade can be done, but it means the ironic services will not be available during the upgrade, which may be time consuming.

Instead of having the services always treat objects in their latest versions, a different design could be used, for example, where pinned services treat their objects in their pinned versions. However, after some experimentation, this proved to have more (corner) cases to consider and was more difficult to understand. This approach would make it harder to maintain and trouble-shoot in the future, assuming reviewers would be able to agree that it worked in the first place!

What if we changed the ironic-api service, so that it had read-only access to the DB and all writes would go via the ironic-conductor service. Would that simplify the changes needed to support rolling upgrades? Perhaps; perhaps not. (Although this author thinks it would be better, regardless, to have all writes being done by the conductor.) With or without this change, we need to ensure that objects are not saved in a newer version (i.e., that is newer than the version in the older release) until all services are running with the new release step 5.2 of the *Rolling upgrade process*. The solution described in this document has objects being saved in their newest versions starting in step 6.1, because it seemed conceptually easy to

understand if we save objects in their latest versions only when a service is unpinned. Wed need a similar mechanism regardless.

Of course, there are probably other ways to handle this, like having all services register what versions they are running in the database and leveraging that data somehow. Dmitry Tantsur mused about whether some remote synchronization stuff (e.g. etcd) could be used for services to be aware of the upgrade process.

Ideally, ironic would use some OpenStack-preferred way to implement rolling upgrades but that doesnt seem to exist, so this tries to leverage the work that nova did.

Data model impact

A DB migration policy is adopted and introduced above in *New DB model change policy*.

A new `version` column will be added to all the database tables of the IronicObject objects. Its value will be the version of the object that is saved in the database.

State Machine Impact

None

REST API impact

There is no change to the REST API itself.

During the rolling upgrade process, the API services may run in different versions at the same time. Both API service versions should be compatible with the python-ironicclient library from the older release (we already use microversions to guarantee this). New API functionality is available everywhere only after completing the upgrade process, which includes unpinning of the internal RPC communication versions on all ironic services. Therefore, until the upgrade is completed, API requests should not be issued for new functionality (i.e., with new API micro versions) since there is no guarantee that they will work properly.

As a future enhancement (outside the scope of this specification), we could disallow requests for new functionality while the API service is pinned.

Client (CLI) impact

None

ironic CLI

None

openstack baremetal CLI

None

RPC API impact

There is no change to the RPC API itself, although changes are needed for supporting different RPC API versions. `version_cap` will be set to the pinned version (the previous release) to make compatible RPC calls of the previous release.

Developers should keep this in mind when changing an existing RPC API call.

Driver API impact

None

Nova driver impact

Since there is no change to the REST API, there is no need to change the nova driver. Ironic should be upgraded before nova, and nova calls ironic using a specific micro version that will still be supported in the upgraded ironic. Thus everything will work fine without any changes to the nova driver.

Ramdisk impact

None

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

Operations which involve migration of data may take longer during the upgrade. Each such change should be mentioned in the release notes provided with the impacted release.

Other deployer impact

During the rolling upgrade, the deployer will use the new configuration option `[DEFAULT]/pin_release_version` to pin and unpin the RPC and IronicObject versions used by the services, as described above in *RPC and object version pinning*.

This specification doesnt address trying to stop/roll back to a previous release.

Developer impact

The code contribution guide[11] will be updated to describe what a developer needs to know and follow. It will mention:

- before a release is cut, manually add the mapping of release (named or sem-versioned) with the associated RPC and Object versions
- the new DB model change policy
- the contribution guide will point to design documentation, as it relates to how new features should be implemented in accordance with rolling upgrades

[11] http://docs.openstack.org/developer/ironic/dev/code-contribution-guide.html#live-upgrade-related-concerns

Implementation

Assignee(s)

Primary assignee:

- xek
- rloo

Other contributors:

- mario-villaplana-j (documentation)

Work Items

1. Add new configuration option `[DEFAULT]/pin_release_version` and RPC/Object version mappings to releases.
2. Make Objects compatible with a previous version and handle the interaction with DB and services.
3. Make IronicObjectSerializer downgrade objects when passing via RPC.
4. Add tests.
5. Add documentation and pointers for RPC and oslo objects versioning.
6. Add documentation for the new DB model change policy.
7. Add admin documentation for operators, describing how to do a rolling upgrade, including the order in which all related services should be upgraded.

Dependencies

- Needs the new command **ironic-dbsync online_data_migration**[Page 1449, 12].
- Needs multi-node grenade CI working.

Testing

- unit tests
- multi-node grenade CI. This tests that the rolling upgrade process continues to work from `fromVer` to `toVer`.
  - grenade does a full upgrade, without pinning. This tests old API/conductor and new API/conductor unpinned. We wont run online data migrations, so the new services will be reading the old format of the data.
  - grenade multi-node will be running old API/conductor on the subnode, which wont be upgraded. The primary will have conductor only, which does get upgraded, but is pinned. This tests old API + old data, with 1. old conductor, and 2. new conductor with a pin. Tests are run pre/post-upgrade.
  - We could also move the API to the primary node, upgrade it, pinned, to test new API code with the pin. Since all the object translation happens at the objects layer, this may not be needed since it may not test much that hasnt already been exercised in the conductor.
  - The above two tests could be merged, if grenade can be set-up to stop the old API service on the subnode and start it on the upgraded primary after doing the first test.

Tests should cover the use cases described in *Rolling upgrades between releases* as much as possible.

Upgrades and Backwards Compatibility

None; there is no change to the REST API or Driver API.

Documentation Impact

Documentation will be added for:

- deployers. This will describe the rolling upgrade process and the steps they will need to take. This should be documented or linked from the upgrade guide[Page 1447, 13].
- developers. This will describe what a developer needs to know so that the rolling upgrade process continues to work. This includes documentation on RPC and oslo objects versioning, as well as DB model change policy.

References

5.15.200 Smart NIC Networking

https://storyboard.openstack.org/#!/story/2003346

This spec describes proposed changes to Ironic to enable a generic, vendor-agnostic, baremetal networking service running on smart NICs, enabling baremetal networking with feature parity to the virtualization use-case.

Problem description

While Ironic today supports Neutron provisioned network connectivity for baremetal servers through an ML2 mechanism driver, the existing support is based largely on configuration of TORs through vendor-specific mechanism drivers, with limited capabilities.

Proposed change

There is a wide range of smart/intelligent NICs emerging on the market. These NICs generally incorporate one or more general purpose CPU cores along with data-plane packet processing acceleration, and can efficiently run virtual switches such as OVS, while maintaining the existing interfaces to the SDN layer.

The proposal is to extend Ironic to enable use of smart NICs to implement generic networking services for Bare Metal servers. The goal is to enable running the standard Neutron Open vSwitch L2 agent, providing a generic, vendor-agnostic bare metal networking service with feature parity compared to the virtualization use-case. The Neutron Open vSwitch L2 agent manages the OVS bridges on the smart NIC.

In this proposal, we address two use-cases:

1. Neutron OVS L2 agent runs locally on the smart NIC.

   This use case requires a smart NIC capable or running openstack control services such as the Neutron OVS L2 agent. This use case strives to view the smart NIC as an isolated hypervisor for the baremetal node, with the smart NIC providing the services to the bare metal image running on the host (as a hypervisor would provide services to a VM). While this spec initially targets Neutron OVS L2 agent, the same implementation would naturally and easily be extended to any other ML2 plugin as well as to additional agents/services (for example exposing emulated NVMe storage devices back-ended by a storage initiator on the smart NIC).

2. Neutron OVS L2 agent(s) run remotely and manages the OVS bridges for all the baremetal smart NICs.

The enhancements for Neutron OVS L2 agent captured in[1],[2] and[3].

- Set the smart NIC configuration

  smart NIC configuration includes the following:

  1. extend the ironic port with is_smartnic field. (default to False)
  2. smart NIC hostname - the hostname of server/smart NIC where the Neutron OVS agent is running. (required)
  3. smart NIC port id - the port name that needs to be plugged to the integration bridge. B in the diagram below (required)
  4. smart NIC SSH public key - ssh public key of the smart NIC (required only for remote)
  5. smart NIC OVSDB SSL certificate - OVSDB SSL of the OVS in smart NIC (required only for remote)

  The OVS ML2 mechanism driver will determine if the Neutron OVS Agent runs locally or remotely based on smart NIC configuration passed from ironic. The config attribute will be stored in the local_link_information of the baremetal port.

  In the scope of this spec the smart NIC config will be set manually by the admin.

- Deployment Interfaces

  Extending the ramdisk, direct, iscsi and ansible to support the smart nic use-cases.

  The Deployment Interfaces call network interface methods such as: add_provisioning_network, remove_provisioning_network, configure_tenant_networks, unconfigure_tenant_networks, add_cleaning_network and remove_cleaning_network.

  These network methods are currently ordinarily called when the baremetal is powered down, ensuring proper network configuration on the TOR before booting the bare metal.

  smart NICs share the power state with the baremetal, requiring the baremetal to be powered up before configuring the network. This leads to a potential race where the baremetal boots and access the network prior to the network being properly configured on the OVS within the smart NIC.

  To ensure proper network configuration prior to baremetal boot, the deployment interfaces will intermittently boot the baremetal into the BIOS shell, providing a state where the ovs on the smart NIC may be configured properly before rebooting the bare metal into the actual guest image or ramdisk. The ovs on the smart NIC will get programmed after we verify that the neutron ovs agent is alive.

  The following code for configure/unconfigure network:

```
if task.driver.network.need_power_on(task):
    old_power_state = task.driver.power.get_power_state(task)
    if old_power_state == states.POWER_OFF:
        # set next boot to BIOS to halt the baremetal boot
        manager_utils.node_set_boot_device(task, boot_devices.BIOS,
```

(continues on next page)

[1] https://review.opendev.org/#/c/619920/
[2] https://review.opendev.org/#/c/595402/
[3] https://review.opendev.org/#/c/595512/

(continued from previous page)

```
                                                persistent=False)
        manager_utils.node_power_action(task, states.POWER_ON)

# ...
# call task.driver.network method(s)
# ...

if task.driver.network.need_power_on(task):
    manager_utils.node_power_action(task, old_power_state)
```

The following methods in the deployment interface are calling to one or more configure/unconfigure networks and should be updated with the logic above.

  - iscsi Deploy Interface
    - iscsi_deploy::prepare
    - iscsi_deploy::deploy
    - iscsi_deploy::tear_down
  - ansible Deploy Interface
    - ansible/deploy::reboot_and_finish_deploy
    - ansible/deploy::prepare
    - ansible/deploy::tear_down
    - ansible/deploy::prepare_cleaning
    - ansible/deploy::tear_down_cleaning
  - direct Interface
    - agent::prepare
    - agent::tear_down
    - agent::deploy
    - agent::rescue
    - agent::unrescue
    - agent_base_vendor::reboot_and_finish_deploy
    - agent_base_vendor::_finalize_rescue
  - RAM Disk Interface
    - pxe::deploy
  - Common cleaning methods
    - deploy_utils::prepare_inband_cleaning
    - deploy_utils::tear_down_inband_clean

- Network Interface

  Extend the base *network_interface* with need_power_on - return true if any ironic port attached to the node is a smart nic

Extend the ironic.common.neutron add_ports_to_network/ remove_ports_from_network methods for the smart NIC case:

- on add_ports_to_network and has smartNIC do the following:
  - check neutron agent alive - verify that neutron agent is alive
  - create neutron port
  - check neutron port active - verify that neutron port is in active state
- on remove_ports_from_network and has smartNIC do the following:
  - check neutron agent alive - verify that neutron agent is alive
  - delete neutron port
  - check neutron port is removed

- Neutron ml2 OVS changes:
  - Introduce a new vnic_type for `smart-nic`.
  - Update the Neutron ml2 OVS to bind smart-nic vnic_type with *binding:profile* smart NIC config.
- Neutron OVS agent changes:

Example of smart NIC model:

```
+--------------------+
|      baremetal     |
| +----------------+ |
| |  OS Server     | | |
| |                | | |
| |      +A        | | |
| +------|--------+ | |
|        |          | |
| +------|--------+ | |
| |  OS SmartNIC   | | |
| |    +-+B-+      | | |
| |    |OVS |      | | |
| |    +-+C-+      | | |
| +------|--------+ | |
+--------|-----------+
         |


A - port on the baremetal host.
B - port that represents the baremetal port in the smart NIC.
C - port that represents to the physical port in the smart NIC.

Add/Remove Port B to the OVS br-int with external-ids

In our case we will use the neutron OVS agent to plug the port on update
port event with the following external-ids: iface-id,iface-status, attached-
↪mac
and node-uuid
```

Alternatives

- Delay the Neutron port binding (port binding means setting all the OVSDB/Openflows config on the SmartNIC) to be performed by Neutron later (once the bare metal is powered up). The problem with this approach is that we have no guarantee of if/when the rules will be programmed, and thus may inadvertently boot the baremetal while the smart NIC is still programmed on the old network.

Data model impact

A new `is_smartnic` boolean field will be added to Port object.

State Machine Impact

None

REST API impact

The port REST API will be modified to support the new `is_smartnic` field. The field will be readable by users with the baremetal observer role and writable by users with the baremetal admin role.

Updates to the is_smartnic field of ports will be restricted in the same way as for other connectivity related fields (link local connection, etc.) - they will be restricted to nodes in the `enroll`, `inspecting` and `manageable` states.

Client (CLI) impact

ironic CLI

None

openstack baremetal CLI

The openstack baremetal CLI will be updated to support getting and setting the `is_smartnic` field on ports.

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

None

Security impact

- Smart NIC Isolation

Both use cases run infrastructure functionality on the smart NIC, with the first use case also running control plane functionality.

This requires proper isolation between the untrusted bare metal host and the smart NIC, preventing any/all direct or indirect access, both through the network interface exposed to the host and through side channels such as the platform BMC.

Such isolation is implemented by the smart NIC device and/or the hardware platform vendor. There are multiple approaches for such isolation, ranging from completely physical disconnection of the smart NIC from the platform BMC to a platform with a trusted BMC wherein the BMC considers the baremetal host an untrusted entity and restricts its capabilities/access to the platform.

In the absence of such isolation, the untrusted baremetal tenant may be able to gain access to the provisioning network, and in the second may be able to compromise the control plane.

Proper isolation is dependent on the platform hardware/firmware, and cannot be directly enforced/guaranteed by ironic. Users of smart NIC use case should be made well aware of this via explicit documentation, and should be guided to verify the proper isolation exists on their platform when enabling such use cases.

- Security Groups

This will allow to use Neutron OVS agent pipeline. One of the features in the pipeline is security groups which will enhance the security model when using baremetal in a cloud.

- Security credentials

The node running the Neutron OVS agent (smart NIC or remote, according to use case) should be configured with the message bus credentials for the Neutron server.

In addition, for the second use case, the SSH public key and OVSDB SSL certificate should be configured for the smart NIC port.

Other end user impact

- Baremetal admin needs to update the SmartNIC config manually.

Scalability impact

None

Performance Impact

None

Other deployer impact

None

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
hamdyk - hamdy@mellanox.com

Work Items

- Update the Neutron network interface to populate the Smart NIC config from the ironic port to the Neutron port *binding:profile* attribute.
- Update the network_interface and common.neutron as described above
- Update deployment interfaces as described above
- Documentation updates.

Dependencies

None, but the Neutron specs[Page 1458, 1],[Page 1458, 2] and[Page 1458, 3] depend on this spec.

Testing

- Mellanox CI Jobs testing with Bluefield SmartNIC

Upgrades and Backwards Compatibility

None

Documentation Impact

- Update the multitenancy.rst with setting the SmartNIC config
- Document the security implications/guidelines under admin/security.rst

References

5.15.201 Synchronize events with Neutron

https://storyboard.openstack.org/#!/story/1304673

Most of neutron operations are asynchronous. We need to keep track of their results by leveraging neutrons notifications, that it can send on certain events to ironic via HTTP calls.

Problem description

Updates to Neutron resources via its API are processed asynchronously on its backend. This exposes potential races with Ironic.

Example: an API request from Ironic to update a ports DHCP settings will return successfully long before the associated dnsmasq config has been updated and the server restarted. There is a potential for a race condition where ironic will boot a machine before its DHCP has been properly configured, especially if the machine boots very quickly (e.g. a local VM).

Another issue, that has a more serious security impact, is if Neutron failed to bind the port when configuring the tenant network, ironic proceeds with finishing the deployment, leaving the port bound to the provisioning network.

Ironic should be able to receive notifications from neutron when the port state is changed. Only Ironic related ports should cause neutron to send notifications. This spec concentrates on the binding operations, but the event handlers can be extended further, for example to be able to process updates to DHCP options. Here is an example of nova notifier in neutron[1].

In this spec, the term notification is used when talking about the notifications sent by neutron. The term event is more generic, and is used when talking about the payload that is received on the ironic HTTP API.

Proposed change

When ironic changes neutron port information during provisioning or cleaning (e.g. updates port binding) or creates a port (multitenancy integration), we put the node into a `*WAIT` state, and pause the deployment/cleaning waiting for neutron notification. As several ports can be updated simultaneously, we need to put the node into a `*WAIT` state only after sending the requests for all of them.

The notifications are generated on neutron side, and are sent to a dedicated ironic API endpoint - `/events`. The framework for this is already present, there is no need for additional work on neutron side. Nova notifier can be seen at[1]. An external network event will contain the following fields:

```
{
    "events": [
        {
            "event": "network.bind_port"
            "port_id": "VIF UUID",
            "mac_address": "VIF MAC address",
            "status": "VIF port status",
            "device_id": "VIF device ID",
            "binding:host_id": "hostname",
        },
        ...
    ]
}
```

Event handler is an object processing the events received on the `/events` endpoint. This spec handles the neutron notifications case, but to make it more generic, it is proposed to define the event handlers inside the related driver interface, in the interface class `event_handler` field. For example, for network related events, it is going to be the following:

```
class NeutronNetwork(common.VIFPortIDMixin,
                     neutron.NeutronNetworkInterfaceMixin,
                     base.NetworkInterface):
    """Neutron v2 network interface"""

    event_handler = NeutronEventHandler()
```

The base `BaseNetworkEventHandler` class will contain the following methods:

[1] https://github.com/openstack/neutron/blob/master/neutron/notifiers/nova.py

```
class BaseNetworkEventHandler(object):

    @abc.abstractmethod
    def configure_tenant_networks(self, task):
        """Ensures that all tenant ports are ready to be used."""
        pass

    @abc.abstractmethod
    def unconfigure_tenant_networks(self, task):
        """Ensures that all tenant ports are down."""
        pass

    @abc.abstractmethod
    def add_provisioning_network(self, task):
        """Ensures that at least one provisioning port is active."""
        pass

    @abc.abstractmethod
    def remove_provisioning_network(self, task):
        """Ensures that all provisioning ports are deleted."""
        pass

    @abc.abstractmethod
    def add_cleaning_network(self, task):
        """Ensures that at least one cleaning port is active."""
        pass

    @abc.abstractmethod
    def remove_cleaning_network(self, task):
        """Ensures that all cleaning ports are deleted."""
        pass
```

In the conductor methods that deal with network interface (e.g., do_node_deploy), were going to be saving the events we are expecting in the nodes `driver_internal_info['waiting_for']` field and calling the network interface methods (this will be moved to conductor from the deploy interface). If the call is synchronous, well just proceed to the next one when the previous is done, otherwise well be triggering the state machine wait event and saving the callback that will be called when the corresponding event is done. All callbacks will be stored in a simple dictionary keyed by node UUID. `driver_internal_info['waiting_for']` is going to be a simple list of strings, each of which is going to be the corresponding driver interface and event handlers method name, so that we know which method of the event handler of a specific driver interface to trigger when an action is asynchronous and we receive the event. If an unexpected event is received, well be ignoring it (and logging that an unexpected event appeared in the API).

Third-party driver interface methods can be also adding things they want to wait for by:

- adding event names into the `driver_internal_info['waiting_for']` list;
- adding event names to callback mappings into global per-node `CALLBACKS` dictionary, along with the arguments with which it should be called.

This will allow to wait for custom events registered in custom driver interfaces.

Neutron does not know which method name it should include in the request body, as it only operates on the neutron entities, it knows about things such as port bound, port unbound, port deleted etc. We will be mapping the things were waiting for to things neutron passes in via the simple dictionary:

```
NETWORK_HANDLER_TO_EVENT_MAP = {
    'network.unconfigure_tenant_networks': 'network.unbind_port',
    'network.configure_tenant_networks': 'network.bind_port',
    'network.add_provisioning_network': 'network.bind_port',
    'network.remove_provisioning_network': 'network.delete_port',
    'network.add_cleaning_network': 'network.bind_port',
    'network.remove_cleaning_network': 'network.delete_port',
}
```

When an external network event is received, and if were waiting for it, ironic API performs node-by-mac and port-by-mac lookup, to check that the respective node and port exist. The port status received in the request body is saved to the ports `internal_info['network_status']`, and then `process_event` is triggered. On the conductor side, `process_event` will be doing the event name to event handler method translation via `NETWORK_HANDLER_TO_EVENT_MAP`, and calling the event handler. Conductor will also be dealing with state machine transitions.

The event handler will be looking at the status of the ironic resources, for example, in case of network events, we want to save the neutron port status in each port or port group to `internal_info['network_status']` and consider an asynchronous action done only when port(group)s have the desired status. The event handler method that needs to be called on the event retrieval should be present in the event body generated by neutron. In case of desired event is done, it should be removed from the `driver_internal_info['waiting_for']` list, and the provisioning action can proceed, by triggering the continue state machine event and calling the callback that we have saved before.

To ensure that we dont wait for events forever, the usual `*WAIT` states timeout periodic tasks will be used. A new one will be added for the new `DELETE WAIT` state. An example of such periodic task is at[2].

Alternatives

- Using semaphores to pause the greenthread while waiting for events. This will make the code clearer and simpler, with only one downside if the conductor is going to be restarted, well loose the info about the events we wait for. This is still better than what we have now, and possibly can be worked around. Another downsides here being possible performance issues if a lot of greenthreads are running sumiltaneously, and the fact that conductor goes down during the rolling upgrade.
- Use Neutron port status polling. There is an issue with that, as even if the neutron ports status is ACTIVE, some actions might not have finished yet. The neutrons notifier framework handles this problem for us.

Data model impact

None.

[2] https://github.com/openstack/ironic/blob/f16e7cdf41701159704697775c436e9b7ffc0013/ironic/conductor/manager.py#L1458-L1479

State Machine Impact

A new `DELETE WAIT` state is introduced. Nodes can move to it from `DELETING` state, upon receiving wait event. When continue event is triggered while the node is in `DELETE WAIT`, the node switches back to the `DELETING` state. This is introduced because we need to unconfigure the tenant networks prior to starting the cleaning.

REST API impact

The new endpoint `POST /events` needs to be created. The default policy for this endpoint will be `"rule:is_admin"`. Request body format is going to be the following:

```
{
    "events": [
        {
            "event": "network.bind_port"
            "port_id": "VIF UUID",
            "mac_address": "VIF MAC address",
            "status": "VIF port status",
            "device_id": "VIF device ID",
            "binding:host_id": "hostname",
        },
        ...
    ]
}
```

Only `event` field is required, and it has the format of `<event_type>.<event>`, where:

- `<event_type>` is a name of the interface whose event handler will be called, during this spec implementation only `network` interface handlers will be added.
- `<event>` is a name of the event that has happened, it will be converted to the event handler method name of the current `<event_type>` interface handler that will be called.

If the expected event handling fails, `fail` state machine event is triggered by the conductor.

The normal response code to the request on this endpoint is 200 (OK), the error codes are:

- 400 (Bad Request), in case of none of the event handlers can process the event.
- 404 (Not Found), in case of making a request with old API microversion header, or if the node can not be found by the MAC address that is sent in the request body.
- 401 (Unauthorized), if the authorization has been refused for the provided credentials.
- 403 (Forbidden), if the user that has issued the request is not allowed to use this endpoint.

Client (CLI) impact

Client will be updated to support sending an external notification. This functionality will only be added to the clients python API, no new commands are going to be introduced. The new method will just be passing the JSON it receives to the `/events` endpoint.

This method will be used by the ironic notifier module within neutron to send the notification to the ironic API.

RPC API impact

A new method `process_event` is going to be added. Received external event is processed here.

In the conductor side of this method we compare current event were waiting for stored in `driver_internal_info['waiting_for']` field with received `"event"` in the event body. If we received the desired event for all port(group)s we need, we trigger `continue` event on the state machine, and the callback that was saved into the per-node `CALLBACKS` dictionary prior to triggering the state machines wait is called.

Driver API impact

As part of this change, to ensure that the network interface calls happen, and we wait for their completion, well need to make the `add_{cleaning,provisioning}_network` network interface methods idempotent, so that we can call them in the conductor without breaking the out-of-tree network interfaces.

Nova driver impact

None.

Security impact

With the move of the network interface calls to conductor, and waiting for their successful completion, we ensure that the network configuration corresponds to what we expect, thus enhancing security, and getting rid of bugs with giving an instance to a user that is still mapped to provisioning network if `remove_provisioning_network` and `configure_tenant_networks` methods fail asynchronously for that port.

Other end user impact

The neutron notifier needs to be configured. It needs the keystone admin credentials and (optionally, if not provided will be discovered from keystone endpoint catalog) the ironic API address to send events to.

Scalability impact

None.

Performance Impact

The node provisioning and unprovisioning may take some additional time when well be waiting for the external events.

Other deployer impact

None.

Ramdisk impact

None.

Developer impact

Developers will be able to create the needed event handlers for whatever events they would like to use during provisioning, and add those to the driver interfaces.

Implementation

Assignee(s)

Primary assignee:
vdrok

Other contributors:
None

Work Items

1. Add the event handlers to neutron and flat network interfaces.
2. Add the `process_event` conductor method, that will be handling the events.
3. Add the `/events` endpoint.
4. Implement the client side of changes.

Dependencies

- Nova should be cleaning up only the ports owned by compute in case of the InstanceDeployFailure[3].

Testing

Integration and unit testing will be provided.

Upgrades and Backwards Compatibility

This does not affect the usual upgrade procedure. To make use of events, both API and conductor need to be upgraded. During upgrade, the ironic notifier needs to be configured in neutron. There is going to be no need to enable this feature, it will be enabled by default.

In case of rolling upgrade, ironic conductors are upgraded first, then ironic APIs, then neutron is reconfigured to enable the notifier.

If we decide to make all the network interface calls asynchronous, step of enabling notifier in neutron becomes obligatory, otherwise an operator will have to send the notifications to ironic API manually, or the deployment will be failing by timeout as no network event is going to be received. This bit might need to be revisisted during review :)

Documentation Impact

This feature will be documented in the developer documentation and API reference.

[3] https://bugs.launchpad.net/nova/+bug/1673429

References

5.15.202 Third Party Driver Continuous Integration Testing

https://bugs.launchpad.net/ironic/+bug/1526410

This spec is to define the deadlines, requirements and process of bringing third party continuous integration testing to ironic.

Problem description

The ironic project wants to make sure that all the drivers in tree are maintained and of high quality. In order to do this, it has become necessary to require that all drivers have a continuous integration test system to test the driver against every code change proposed for ironic, unless that change is to code or documentation that can not impact the driver. Required tests to be run by each driver test system will be documented in the ironic CI requirements documentation. Not all drivers are provided or maintained by third party vendors.

Third party vendors must provide and maintain this CI for drivers they maintain. Any driver whose maintainer is not able to implement a reliable CI system to test will be removed from the ironic tree.

A driver test system will be deemed reliable if it runs the expected tests and reports the results of those tests consistently over a period of time. Tests will be expected to complete and report back to gerrit within 8 hours of the patch submission until the end of Newton release and within 4 hours by the end of Ocata development cycle.

A driver test system will be deemed reliable if:

- It runs the expected tests for every patch set that is not excluded. Reasons for this exclusion will be documented and approved by the ironic team.
- It reports the results within the expected time frame, see above.
- All required test artfacts are posted in a standard format and made available to the community following the infrastructure requirements.[3]
- Test results (pass or fail) are accurate.

Proposed change

The ironic team has decided to require all driver maintainers to run and maintain CI test systems in order to have their driver code remain in the ironic source tree. When there are problems with the CI system, the driver test team will make a best effort to respond to questions about their system and to fix the system quickly. If the driver test team is not responsive, any voting privileges for the test system may be removed, and may eventually result in the driver being removed from the source tree.

Timeline

The process to implement and start enforcing third party driver CI is to be completed by the end of the Newton development cycle.

Deliverable milestones:

- Immediately: Driver teams contacted and notified of expectations.
- Mitaka-2 milestone: Driver teams will have registered their intent to run CI by creating system accounts and identifying a point of contact for their CI team.

- Mitaka Feature Freeze: All driver systems show the ability to receive events and post comments in the third party CI sandbox.
- N Feature Freeze: Per patch testing and posting comments.

Please refer to the Mitaka release schedule[5] for specific dates. Note that the ironic project does not strictly follow the official OpenStack release cycle deadlines, but we are using the official deadlines as a reference point.

Infra CI will continue to test the ssh drivers already being used in the gate.

Third party driver teams that do not implement a reliable reporting CI test system by the Newton release feature freeze (see Deliverable milestones above) will be removed from the ironic source tree. Driver test systems that miss any of the milestones may be subject to immediate removal from the source tree.

Driver test systems will be required to initially run a test similar to the *gate-tempest-dsvm-ironic-pxe_ipa* test, with the only difference being changes to drivers loaded, etc, to make ironic work with the driver under test.

More information about this and other required tests will need further documentation.

Initial driver test systems will not be allowed to vote on changes to ironic. A driver test team may ask that their CI system be allowed to vote only if the ironic team approves based on the test teams participation, availability and history of reliable operation.

Driver test systems will be expected to follow the Infrastructure Third Party test requirements[3], unless overridden here or in the ironic third party driver testing documentation produced as a result of this spec.

Driver test systems must test patches to master, and patches to branches that were created since the third party CI began testing. For example, if a CI began testing during the Mitaka cycle, then during the N cycle that CI will be expected to test changes to the master and stable/mitaka branches, but not the stable/liberty branch.

Community-maintained drivers that do not have CI testing will also be removed from the ironic source tree.

From Newton feature freeze forward, all new ironic drivers must show that they have a system performing CI testing and meet all infra [3] and ironic requirements in order to land code into the ironic tree. Drivers in progress before Newton feature freeze have until Newton feature freeze to meet these requirements. CI testing does not need to be shown in order to propose a driver spec, but must be in place before landing code.

Alternatives

None

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

N/A

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

When upgrading to the release that drops untested drivers, if a deployer is using a driver that is removed from the tree, they will need to change to an in-tree driver or install the removed driver from a new location, if one exists.

The Ironic team must communicate which drivers are being removed, and when. We should note that these drivers *may* be available at a new location, and that driver authors *may* be communicating that information.

Authors of a driver removed from tree may communicate the new location, if one exists, and document how to install the driver into an ironic environment.

Developer impact

Developer impacts may include core reviewers needing to wait until testing for a system completes before approving a patch for merge. Developers that had a test fail will need to review the test artifacts for their patch linked to the comment left in the patch comment log. If necessary, the developer may need to coordinate with the driver test team for help with debugging the problem. See Infra requirements in the References section below.

Implementation

Assignee(s)

Primary assignee:
krtaylor

Other contributors:
jroll, thingee

Work Items

1. Communicate intention to vendors with existing drivers in tree - make a reasonable effort to contact the entity responsible for the driver and inform them of the timeline to require driver third party CI.
2. Set incremental timeline milestones for vendors to implement CI testing.
3. A deprecation process will need to be documented.
4. Document process, requirements - this spec is not meant to exhaustively enumerate all requirements, just to define that they need to be documented.
5. The documentation will also need to describe the way in which a test system proves they are adequately testing their driver.
6. Assemble and maintain list of contacts for all in-tree drivers.
7. Remove third party drivers that do not implement a CI test system as per the schedule listed above.
8. Document impacts to ironic deployers and developers that the driver they may have been using was removed from tree, as per the deployer impact section.

Dependencies

None

Testing

As described in this spec.

Upgrades and Backwards Compatibility

There will be a major upgrade impact on deployers using drivers that are removed from tree; see the Deployer impact section for more info.

A deprecation process will be documented including timeline.

Documentation Impact

There will be several areas impacted:

1. Document drivers in tree and their expected functionality.
2. Document requirements for the third party drivers systems, expectations, time thresholds, tests required to be run, and other topics as needed.
3. Document an example implementation of the third party test system infrastructure.
4. Document the process to notify the community and users that a driver will be removed from tree.
5. Document more information about the required tests

References

[1] Third Party CI working group https://wiki.openstack.org/wiki/ThirdPartyCIWorkingGroup

[2] Third party CI meetings https://wiki.openstack.org/wiki/Meetings/ThirdParty

[3] Infra requirements documentation for implementing a third party system http://docs.openstack.org/infra/system-config/third_party.html#requirements

[4] Discussion at Mitaka summit https://etherpad.openstack.org/p/summit-mitaka-ironic-third-party-ci

[5] Mitaka release schedule: https://wiki.openstack.org/wiki/Mitaka_Release_Schedule

5.15.203 Client Caching Of Negotiated Version

https://bugs.launchpad.net/ironic/+bug/1526411

This adds support for caching the version negotiated by the ironicclient, between itself and the ironic server. This is supplementary to the api microversion spec approved in the Kilo release[0].

Problem description

When the ironicclient talks to the ironic server, there may be a mismatch in supported API versions. The client and server negotiate a version to be used in communicating, but since the ironicclient can be used as a user interactive client, this process of negotiation would be repeated for each command line invocation.

It would be useful, for each ironic server that the ironicclient talks to, to cache the version agreed upon for communication, so that each conversation between client and server does not require renegotiation.

This version caching would need to be per user, for each ironic server (host, network port pair specific) and would be time-bound (say 5 minutes), so any future upgrade of the client or server would benefit from supporting newer available versions.

Proposed change

The proposed implementation consists of caching the negotiated version information between an ironic-client and ironic server in local file storage for use by future invocations of the ironicclient.

This information would be cached for a specific period of time, before becoming stale and ignored.

Specifically, we are proposing:

- Using the dogpile.cache[1] caching system, a pre-existing library that is already included in global requirements[2]. It is currently used by os-client-config[3], which is used by openstackclient[4].

- Storing the cached information in local file storage, using appdirs[5] to provide the correct location
- Indexing the version information in an ironic-server:network-port pair (such as example.com:1234) so that multiple ironic servers running on the same IP address will be cached independently for each user invoking python-ironicclient
- Having a default period of 5 minutes for caching version information for each ironic server

Storing the version information in a well-known, standardised, local file storage location means that, if the user wants to, they can remove the cached version information manually triggering a renegotiation of the version to be used in communication between the client and server.

Alternatives

An alternative file-based solution was proposed[6], but rejected in favour of using dogpile.cache.

The suggestion to move to dogpile.cache was made both in the code review[9] and was discussed in IRC[8].

Reasons for using dogpile.cache included: commonality with existing file caching libraries used elsewhere in OpenStack, and use of tested common libraries typically means less bugs.

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

This spec only affects the python-ironicclient, not ironic server.

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

N/A

Security impact

None

Other end user impact

None

Scalability impact

This change potentially reduces network traffic between the client and server and hence aids scalability.

Performance Impact

This change potentially reduces network traffic between the client and server and hence improves latency between when a request is made to ironic and when the response is received.

Other deployer impact

None

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
mrda - Michael Davies <michael@the-davies.net>

Work Items

- The implementation of this spec has already commenced - see [7]

Dependencies

None

Testing

Unit tests will be provided to verify this solution

Upgrades and Backwards Compatibility

None

Documentation Impact

None

References

- [0] API Microversions Spec: http://specs.openstack.org/openstack/ironic-specs/specs/kilo/api-microversions.html
- [1] Documentation on dogpile.cache is found here: https://dogpilecache.readthedocs.org/en/latest/
- [2] dogpile.cache is already specified in https://github.com/openstack/requirements/blob/master/global-requirements.txt
- [3] https://github.com/openstack/os-client-config
- [4] https://github.com/openstack/python-openstackclient
- [5] Documentation on appdirs is found here: https://pypi.org/project/appdirs
- [6] Original custom file cache solution: https://review.opendev.org/#/c/173674/1/
- [7] Current state of the implementation at the time of this spec being raised: https://review.opendev.org/#/c/173674/19
- [8] http://eavesdrop.openstack.org/irclogs/%23openstack-ironic/%23openstack-ironic.2015-05-11.log.html#t2015-05-11T19:38:04
- [9] https://review.opendev.org/#/c/173674/9

5.15.204 Virtual Bare Metal Cluster Manager

StoryBoard RFE

Testing Ironic has always been challenging. Bare metal compute resources are expensive and inflexible. The typical solution to this issue is to emulate bare metal compute resources using virtual machines (VMs), allowing us to improve the utilisation of test infrastructure, and test various scenarios using general purpose test infrastructure. Support for creating virtual bare metal environments is available in DevStack, and also in Bifrost. These are used in OpenStack CI. This specification covers creation of a new tool for managing virtual bare metal clusters.

Problem description

As Ironic and bare metal cloud use cases grow in popularity, it becomes necessary to ensure that tooling is available for performing representative tests. The DevStack plugin and Bifrost playbooks are helpful, but not applicable in every environment, as they pull in additional dependencies, and are quite tied to the specifics of those environments.

In particular, testing of OpenStack deployment tools that make use of Ironic such as Kayobe, Kolla Ansible and TripleO would benefit from a shared tool for managing the configuration of virtual bare metal testing environments.

It would be desirable for the new tool to be extensible and support more complex configuration in future - for example, networking configuration. Shell script-based tools such as the DevStack plugin inherently risk becoming monolithic with new feature additions. It is important that the new tool supports more complex use cases without sacrificing its ease of use or maintenance for simpler configurations.

User Stories

- As an OpenStack developer, I want to be able to easily create a generic all-in-one virtual bare metal cluster at will, so that I can use it as a development environment.
- As an OpenStack developer, I want to be able to tear down my virtual cluster once I am finished with it, so that my system is brought back to a clean state.
- As an Ironic developer, I want to be able to create a virtual bare metal cluster to a particular specification of VMs and virtual networks, so that I can simulate a given system architecture for my own development.
- As an Ironic developer, I want to be able to reconfigure my virtual cluster, so that I can test changes in a variety of bare metal architectures.
- As a maintainer of a CI toolchain, I want there be a *de facto* virtual cluster management tool for CI jobs, so that I dont have to use a bespoke/non-extensible utility.
- As an Ironic developer, I want CI jobs to set up job environments with the same tool that I use to set up my development environment, so that I can easily reproduce the results of CI jobs on my own machine.
- As a systems deployer, I want to be able to perform bare metal scale testing locally/virtually, so I can diagnose scaling issues before deploying onto a real system.

Proposed change

We propose to add a new tool for managing virtual bare metal clusters, named *Tenks*. The scope of Tenks includes:

- Creation and deletion of VMs representing bare metal nodes
- Orchestration of tools used to emulate Baseboard Management Controllers (BMCs), e.g. virtual BMC and *sushy-tools <https://github.com/openstack/sushy-tools>*.
- Creation and deletion of virtual networks representing bare metal networks

Tenks will initially be targeted at the typical Libvirt/QEMU/KVM environment, but in the typical OpenStack way, will be written in such a way as to allow support for other virtualisation providers to be added in future.

N.B. Libvirt provides an extremely extensive API for configuration of VM properties. It is not Tenks aim to expose anything but the smallest common subset of this. This offers the advantages of easier cross-compatibility with multiple virtualisation providers, and looser coupling to each of these. Tenks is intended for development and testing purposes, rather than for fine-grained tuning of virtual cluster configuration.

Once built, we propose to migrate the Ironic DevStack plugin and/or Bifrost test playbooks to use Tenks.

The source code for Tenks is available on GitHub. A blog post about Tenks initial development can be found on the StackHPC website. Documentation can be found on Read the Docs.

Etymological Side Note

- Tenks
- ten-k-s
- 10-thousand-spoons

- Alanis Morissette - Ironic: lyrics L27

Name idea courtesy of mgoddard.

Proposed Implementation

It is proposed to implement Tenks using Ansible. Ansible is a good fit for such a tool because:

- It provides a good platform upon which to write an idempotent application. Since Tenks will be configured declaratively, idempotence is important because it allows the system to be brought to the declared state regardless of its initial state (for example, if some of the declared VMs already exist).
- It allows easy reference to pre-existing Ansible roles to help with configuration.
- It encourages development of an opinioniated tool that can be easily configured as required. This means that Tenks default Ansible configuration would be able to create a basic virtual bare metal cluster with minimal set-up required (due to Ansible role defaults), but any options can be overridden to customise the configuration as required, taking advantage of Ansibles variable precedence and scoping logic.

Dependent on the size that Tenks grows to, it will probably be necessary to write an OpenStack-like CLI to wrap around the Ansible plays.

Multi-Stage Process

- Host setup:
  - Installing and configuring the relevant software on each host:
    - Open vSwitch (containerised?)
    - Virtualisation toolkit (e.g. Libvirt/QEMU/KVM)
    - Virtual platform management toolkit: Virtual BMC for IPMI at the moment. Scope for extension to include other tools in future (e.g. VirtualPDU for SNMP, sushy-tools for Redfish).
    - LVM - set up storage pools for VM volumes
  - Networking setup:
    - Creation of bridges for each physical network, and connection to physical interfaces.
- VM configuration:
  - Schedule requests of different VM specifications to the most appropriate BM hypervisor. The scheduling algorithm could initially be naïve (or even random), and could be iteratively improved. Tenks could even be configured to prioritise certain scheduling heuristics more or less.
  - Create the specified VMs through the virtualisation provider.
  - Attach the VMs NICs to the relevant bridges, based on the physical networks they were configured to be connected to.
  - Register each VM with a platform management tool suitable for its driver (e.g. Virtual BMC for IPMI).
- VM enrolment (these steps are optional if introspection is to be used):

  - Enrol each VM with Ironic, using a specified deployment ramdisk and kernel.
  - Set any additional properties on the node. This could include boot-from-volume details, capabilities and boot mode (for boot modes supported by Tenks).
  - Set any traits on the node.
  - Create a port in Ironic for each of the VMs NICs.
  - Make Ironic nodes available for deployment.
- Post-deployment:
  - Create Nova flavors as required. These can specify node traits that are either desired or forbidden.

Tenks should also support a tear-down mode which would clean up all created resources and restore the system (more or less) to its initial state.

Configuration

A declarative configuration style would be appropriate to describe the virtual infrastructure provisioned by Tenks. This could include:

- Host inventory. Tenks will need a list of bare metal hypervisors, i.e. hosts of virtual machines simulating bare metal nodes. Could use an Ansible inventory for this.
- Physical networks. Tenks would be configured with a list of physical networks that are shared by all hosts in the inventory. A per-host mapping of physical networks to source interfaces/bridges would be required. Tenks would create a bridge for each physical network on each host.
- Desired virtual bare metal VM configuration. Tenks would be configured with flavours of VM, and a count of how many VMs of each flavour to create. Flavours would ideally be agnostic of virtualisation provider, but should have the following properties:
  - Physical networks. A virtual NIC would be added to the VM for each physical network, and the NICs would be plugged into the respective bridge on the hypervisor.
  - Generic VM attributes. These would include number of CPUs and amount of RAM.
  - Volumes to be attached to the VM. Creation of blank volumes and volumes from existing images should be supported, in addition to use of existing volumes.

  These properties could be used to influence VM placement during scheduling. Initial flavour mappings for the Libvirt provider may be facilitated using the StackHPC libvirt-vm Ansible role as an interface.

Alternatives

- Continue using specific tools in each environment

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

None

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

None

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
Will Miller: willm@stackhpc.com

Other contributors:

Work Items

- Develop proof-of-concept Ansible playbooks
- Flesh out Tenks to include more advanced features, for example:
  - Tear-down of clusters
  - Improved scheduling heuristics
  - Reconfiguration of clusters without need for tear-down
  - Providers other than Libvirt
  - Command-line interface
  - Tests (unit, functional, integration) as necessary
- Manual testing of Tenks with various environments/configurations
- Submit Tenks to PyPI (if the extent of Python code requires this)
- Adapt CI pipelines to use Tenks for ephemeral cluster management

Dependencies

None

Testing

TODO

Upgrades and Backwards Compatibility

None

Documentation Impact

TODO

References

- DevStack bare metal network simulation split (abandoned)
- Sam Betts libvirt bare metal simulation
- QuintupleO: OpenStack virtual bare metal

5.15.205 VNC console

https://bugs.launchpad.net/ironic/+bug/1567629

In addition to a serial console, allow ironic nodes to be accessed through a vnc console. This proposal presents the work required to create a new driver interface for accessing graphical console of a node.

Problem description

End users often have to troubleshoot their instances because they might have broken their boot configuration or locked themselves out with a firewall. Keyboard-Video-Mouse (KVM) access is often required for troubleshooting these types of issues as serial access is not always available or correctly configured. Also, KVM provides a better user experience as compared to serial console.

Horizons VNC console is not supported for the ironic nodes provisioned by Nova. This spec intents to extend that to graphical console via the novnc proxy.

The end user will be able to get workable vnc console url from baremetal server: switch console type on bm side to `vnc` `openstack baremetal node console enable` `openstack console url show --novnc`

Proposed change

- In order to support the handshake for VNC authentication we have to implement proxy service as a part of security isolation. During handshake `vnc password` is used. It is stored on ironic side in `driver_info/vnc password` and without proxy need to be provided to Nova. This password should be set by admin. More information about vnc password is in rfb protocol. With novncproxy Nova internals dont need internal details of the BMC network. Expected that this new service can be based on nova_novncproxy.
- for drac will be created a vnc driver based on `base.ConsoleInterface`

Alternatives

- Accept this limitation and only offer a serial console.
- We can configure kvm access including access to the bios via the serial proxy and shell in a box for nova provisioned ironic baremetal instances. This would require exposing credentials.
- Use out-of-band KVM access provided by administrator without Ironic support.

Data model impact

None.

State Machine Impact

None.

REST API impact

None.

Client (CLI) impact

None.

RPC API impact

None.

Driver API impact

None.

Nova driver impact

Nova impacts are fully described in the support of vnc console for ironic spec in Nova.

Essentially, the Ironic virt driver will have to implement `get_vnc_console`

As per policy in Nova, changes cannot land until ironic changes have landed.

Ramdisk impact

None.

Security impact

The VNC connection to the nodes are secured by a token generated while creating the console in Nova. This bearer token is the only thing required to connect to the novnc proxy, So the connection between user and novnc proxy should be protected via ssl

Other end user impact

None.

Scalability impact

None.

Performance Impact

None.

Other deployer impact

additions to configs (idrac example):

`ironic-conductor/ironic.conf`: enabled_console_interfaces = idrac-socat,ipmitool-socat, `idrac-vnc`

`ironic-api/ironic.conf`: enabled_console_interfaces = idrac-socat,ipmitool-socat, `idrac-vnc`

Developer impact

None.

Implementation

Assignee(s)

Primary assignee:

- kirillgermanov

Other contributors:
None.

Work Items

- implement `ironic-novncproxy` service
- Introduce `drac.DracWSManVNCConsole(base.ConsoleInterface)`
- Add usage description to documentation

Dependencies

https://review.opendev.org/c/openstack/nova-specs/+/863773

Testing

- Unit tests

Upgrades and Backwards Compatibility

None.

Documentation Impact

- Documentation will be updated.

References

- https://review.opendev.org/c/openstack/nova-specs/+/863773
- https://stackoverflow.com/questions/16469487/vnc-des-authentication-algorithm
- https://review.opendev.org/c/openstack/ironic/+/860689 - gerrit review ironic
- https://review.opendev.org/c/openstack/nova/+/863177 - gerrit review nova
- https://datatracker.ietf.org/doc/html/rfc6143 - rfb protocol

5.15.206 Add volume connection information for Ironic nodes

https://bugs.launchpad.net/ironic/+bug/1526231

This RFE introduces the changes in Ironic to support connecting and booting instances from remote volumes.

Problem description

When user starts bare metal instance with Cinder volume, Nova orchestrates the communication with Cinder and Ironic. The work flow of the boot process is as follows:

1. (Preparation) Administrator registers a node with initiator information.
2. User asks Cinder to create a boot volume.
3. User asks Nova to boot a node from the Cinder volume.
4. Nova calls Ironic to collect iSCSI/FC initiator information. Ironic collects initiator information and returns it to Nova.
5. Nova calls Cinder to attach the volume to the node. Cinder attaches the volume to the node and returns connection information which includes target information.
6. Nova passes the target information for the node to Ironic
7. Nova calls Ironic to spawn the instance. Ironic prepares the bare metal node to boot from the remote volume which is identified by target information and powers on the bare metal node.

In the work flow above, Nova calls Ironic to get/set initiator/target information (4 and 6) and also administrator calls Ironic to set initiator information (1) but currently Ironic has neither those information nor APIs for them.

Proposed change

- Add a new table named `volume_connectors` with the below fields:
  - id
    - Integer
    - PrimaryKeyConstraint
  - uuid
    - String(length=36)
    - UniqueConstraint
  - node_id
    - Integer
    - ForeignKeyConstraint(nodes.id)
  - created_at
    - DateTime
  - updated_at
    - DateTime
  - type (can have values iqn, ip, mac, wwnn, wwpn, net-id)
    - String(Length=32)
    - UniqueConstraint(type, connector_id)
  - connector_id
    - String(length=255)

        * UniqueConstraint(type, connector_id)
    - extra
        * Text

> **Note**
>
> Ironic should allow users to set IP and/or MAC address hardcorded into *connector_id* field because we cant put any assumptions on the storage network. It might be a Neutron network, or it might be a network that the OpenStack part of the deployment doesnt know about at all. It depends on deployment.

> **Note**
>
> *extra* field is text in the database but a dictionary (JSON-encoded dict) in the object

- Add a new table named `volume_targets` with the below fields:
    - id
        * Integer
        * PrimaryKeyConstraint
    - uuid
        * String(length=36)
        * UniqueConstraint
    - node_id
        * Integer
        * ForeignKeyConstraint(nodes.id)
    - created_at
        * DateTime
    - updated_at
        * DateTime
    - volume_type
        * String(length=64)
    - properties
        * Text
    - boot_index
        * Integer
        * UniqueConstraint(node_id, boot_index)

    - This is used for Ironic to distinguish the root volume. Similar to Nova, Ironic assumes volumes with boot index 0 are root device. (Nova associates a boot index with each block device and assumes volumes with boot index 0 are root volumes.)

  - volume_id
    - String(length=36)
  - extra
    - Text

Note

Ironic should clear the connected target information on node tear_down, just like it does for instance_info.

Note

properties and *extra* are text in the database but a dictionary (JSON-encoded dict) in the object

Note

The contents of the *properties* field depend on volume type. Reference information should be added in Bare Metal API document:

For iSCSI example:

```
{"auth_method": "CHAP",
 "auth_username": "XXX",
 "auth_password": "XXX",
 "target_iqn": "iqn.2010-10.com.example:vol-X",
 "target_portal": "192.168.0.123:3260",
 "volume_id": "12345678-...",
 "target_lun": 0,
 "access_mode": "rw",
 "target_discovered": false,
 "encrypted": false,
 "qos_specs": null}
```

For iSCSI multipath example:

```
{"auth_method": "CHAP",
 "auth_username": "XXX",
 "auth_password": "XXX",
 "target_iqns": ["iqn.2010-10.com.example:vol-X",
 "iqn.2010-10.com.example:vol-Y"],
 "target_portals": ["192.168.0.123:3260",
 "192.168.0.124:3260"],
 "volume_id": "12345678-...",
 "target_luns": [0, 1],
 "access_mode": "rw",
 "target_discovered": false,
 "encrypted": false,
 "qos_specs": null}
```

For fibre channel example:

```
{"device_path": "/dev/disk/by-path/pci-XXXX",
 "encrypted": false,
 "qos_specs": null,
 "target_lun": 1,
 "access_mode": "rw",
 "target_wwn": ["XXXX"]}
```

REST API masks credential information such as *auth_username* and *auth_password* in iSCSI and iSCSI multipath examples in order to avoid security risk.

- Add REST APIs end points to get/set values on them. For details see REST API Impact section.
  - /v1/volume/connectors
  - /v1/volume/targets
  - /v1/nodes/<node_uuid or name>/volume/connectors
  - /v1/nodes/<node_uuid or name>/volume/targets
- Add new capability flags in `node.properties['capabilities']`. These flags show whether or not the node can boot from volume with each backend. If it can boot from volume, we should set the flag to true.
  - iscsi_boot
  - fibre_channel_boot

Note

This should be set to true if the bare metal node supports booting from that specific volume. It might be populated manually by operator or by inspection, but that is not in the scope of this spec.

Note

In the future, Ironic will provide driver capabilities information. Nova can use that information to choose appropriate node.

- If a list of targets are specified, its up to the driver handling the deploy to take care of this. For multi-pathing, Ironic driver, bare metal hardware and the operating system should support it. If Ironic driver and bare metal hardware supports it, but instance operating system doesnt understand it, then it might lead to failure in booting the instance or corrupting the information in the Cinder volume.
- Information which is stored in volume_connector and volume_target tables are used in drivers in order to boot the node from volume. Changes for reference driver, driver interfaces are described in the spec[4].

[4] https://review.opendev.org/#/c/294995

Alternatives

- Saving connector information in a new node attribute like volume_initiator_info. This change has less impact on current code and API but proposed one has more benefits such as better integrity check, faster query from db and easier to store information related to a particular connector.
- Saving target information in a new node attribute like volume_target_info. This change has less impact on current code and API but proposed one has more benefits such as better integrity check, faster query from db and easier to store information related to a particular target.
- Saving target information in instance_info along with other instance related information. This seems to be straightforward because basically target volume information is related to the instance. In this case, `node.instance_info` is nested to store target information. This makes it difficult for users to manipulate target information, and for a driver to validate it. On the other hand, current approach can avoid nesting instance_info and so its easier to use those information. Note, ironic clears the target connection information on the node tear_down.
- Not implement storage of target and initiator information, which ultimately would not improve user experience and require manual post-deployment configuration for out-of-band control. For in-band use, Nova ironic driver can manage initiator information and it is proposed by jroll[2].

Data model impact

- Add new type of object `VolumeConnector` in objects/volume_connector.py. It inherits IronicObject class. The new object will have the following fields:
  - `id`
  - `uuid`
  - `node_id`
  - `type`
  - `connector_id`
  - `extra`
  - `created_at` (defined in IronicObject class)
  - `updated_at` (defined in IronicObject class)
- Add new type of object `VolumeTarget` in objects/volume_target.py. It inherits IronicObject class. The new object will have the following fields:
  - `id`
  - `uuid`
  - `node_id`
  - `volume_type`
  - `properties`
  - `boot_index`
  - `volume_id`
  - `extra`

[2] https://review.opendev.org/#/c/184652/

  - `created_at` (defined in IronicObject class)
  - `updated_at` (defined in IronicObject class)

State Machine Impact

None.

REST API impact

Six new REST API endpoints will be introduced with this change.

- `/v1/volume/connectors`
  - To set the volume connector (initiator) information:

    ```
    POST /v1/volume/connectors
    ```

    with the body containing the JSON description of the volume connector. It will return 201 on success, 400 if some required attributes are missing or having invalid value OR 409 if an entry already exists for the same volume connector.
  - To get information about all volume connectors:

    ```
    GET /v1/volume/connectors
    ```

    This operation will return a list of dictionaries. It contains information about all volume connectors:

    ```
    {
        "volume_connectors":[
            {
                "connector_id": "<wwpn>",
                "links": [ ... ],
                "type": "wwpn",
                "uuid": "<uuid>",
            },
            {
                "connector_id": "<wwpn>",
                "links": [ ... ],
                ...
            },
            ...
        ]
    }
    ```

    This will return 200 on success

    This operation can take parameters like `type`, `container_id`, `limit`, `marker`, `sort_dir`, and `fields`.
  - To get detail information about all volume connectors:

    ```
    GET /v1/volume/connectors/detail
    ```

The operation will return a list of dictionaries. It contains detailed information about all volume connectors:

```
{
    "volume_connectors":[
        {
            "connector_id": "<wwpn>",
            "created_at": "<created_date>",
            "extra": {},
            "links": [ ... ],
            "node_uuid": "<node_uuid>",
            "type": "wwpn",
            "updated_at": "<updated_date>",
            "uuid": "<uuid>",
        },
        {
            "connector_id": "<wwpn>",
            "created_at": "<created_date>",
            ...
        },
        ...
    ]
}
```

It will return 200 on success.

This operation can take parameters like `type`, `container_id`, `limit`, `marker`, and `sort_dir`.

- It should be possible to pass `node` as a parameter which can be a node name or a node UUID to get all volume connectors for that particular node:

```
GET /v1/volume/connectors?node=<node_uuid or name>
GET /v1/volume/connectors/detail?node=<node_uuid or name>
```

It will return 200 on success or 404 if the node is not found.

- `/v1/volume/connectors/<volume_connector_uuid>`
  - To get detail information about a particular volume connector:

```
GET /v1/volume/connectors/<volume_connector_uuid>
```

This will return 200 on success or 404 if volume connector is not found.

  - To update a particular volume connector:

```
PATCH /v1/volume/connectors/<volume_connector_uuid>
```

This will return 200 and the representation of the updated resource on success and 404 if volume connector is not found.

Note

Updating connector information when the node is in POWER_ON or REBOOT state is blocked. It means that users need to make sure the node is in POWER_OFF state before updating connector information. When connector information is updated, driver should update node configuration.

  - To delete volume connector:

```
DELETE /v1/volume/connectors/<volume_connector_uuid>
```

    It will return 204 on success or 404 if volume connector is not found or 400 if the node is not in POWER_OFF state.

- `/v1/nodes/<node_uuid or name>/volume/connectors`
  - To get all the volume connectors information for a node:

```
GET ``/v1/nodes/<node_uuid or name>/volume/connectors``
```

- `/v1/volume/targets`
  - To set the volume target information:

```
POST /v1/volume/targets
```

    with the body containing the JSON description of the volume target. It will return 201 on success, 400 if some required attributes are missing or having invalid value OR 409 if an entry already exists for the same volume target.

  - To get information about all volume targets:

```
GET /v1/volume/targets
```

    This operation will return a list of dictionaries. It contains information about all volume targets:

```
{
    "volume_targets":[
        {
            "boot_index", "<boot_index>",
            "links": [ ... ],
            "uuid": "<uuid>",
            "volume_id": "<volume_id>"
            "volume_type": "<volume_target_type>",
        },
        {
            "boot_index", "<boot_index>",
            "links": [ ... ],
            ...
        },
        ...
    ]
}
```

    This will return 200 on success.

This operation can take parameters like `boot_index`, `volume_id`, `volume_type`, `limit`, `marker`, `sort_dir`, and `fields`.

- To get details information about all volume targets:

```
GET /v1/volume/targets/detail
```

The operation will return a list of dictionaries. It contains detailed information about all volume targets:

```
{
    "volume_targets":[
        {
            "boot_index": "<boot_index>",
            "created_at": "<created_date>",
            "extra": {},
            "links": [ ... ],
            "node_uuid": "<node_uuid>",
            "properties" : { "<target_information>" },
            "updated_at": "<updated_date>",
            "uuid": "<uuid>",
            "volume_id": "<volume_id>",
            "volume_type": "<volume_target_type>",
        },
        {
            "boot_index": "<boot_index>",
            "created_at": "<created_date>",
            ...
        },
        ...
    ]
}
```

It will return 200 on success.

This operation can take parameters like `boot_index`, `volume_id`, `volume_type`, `limit`, `marker`, and `sort_dir`.

Note

properties may include credential information. This API will mask it to avoid security risk.

- It should be possible to pass `node` as a parameter which can be a node name or a node UUID to get all volume targets for that particular node:

```
GET /v1/volume/targets?node=<node_uuid or name>
GET /v1/volume/targets/detail?node=<node_uuid or name>
```

It will return 200 on success or 404 if the node is not found.

- `/v1/volume/targets/<volume_target_uuid>`
  - To get detailed information about a particular volume target:

```
GET /v1/volume/targets/<volume_target_uuid>
```

This will return 200 on success or 404 if volume target is not found.

- To update a particular volume target:

```
PATCH /v1/volume/targets/<volume_target_uuid>
```

This will return 200 and the representation of the updated resource on success, 404 if volume target is not found or 400 if the node is not POWER_OFF state.

Note

Updating target information when the node is in POWER_ON or REBOOT state is blocked. It means that users need to make sure the node is in POWER_OFF state before updating target information. When target information is updated, driver should update node configuration.

- To delete volume target:

```
DELETE /v1/volume/targets/<volume_target_uuid>
```

It will return 204 on success, 404 if volume target is not found or 400 if the node is not in POWER_OFF state.

- `/v1/nodes/<node_uuid or name>/volume/targets`
  - To get all the volume targets information for a node:

```
GET ``/v1/nodes/<node_uuid or name>/volume/targets``
```

- `/v1/nodes/<node_uuid or name>/volume/targets`
  - To get all the volume targets information for a node:

```
GET ``/v1/nodes/<node_uuid or name>/volume/targets``
```

The endpoint `GET /v1/nodes/detail` will provide the volume connectors and targets information for the node with links to them. Also, the endpoint `GET /v1/nodes/<node_uuid or name>` will provide the volume connectors and targets information for the specified node.

For the above REST API changes, micro version will be bumped and 406 will be raised if newer endpoints are accessed with a lesser micro version.

Client (CLI) impact

- A new `VolumeConnectorManager` will be added to `ironicclient` to get/set connector information for the node. Also the CLI will be modified as follows:

```
ironic volume-connector-create --node <node> --type <type>
                               --connector_id <connector_id>
                               [-e <key=value>] [-u <uuid>]
ironic volume-connector-delete <uuid> [<uuid>]
ironic volume-connector-list [--detail] [--type <type>]
```

(continues on next page)

(continued from previous page)

```
                              [--connector_id <connector_id>]
                              [--limit <limit>] [--marker <uuid>]
                              [--sort-key <field>] [--sort-dir <direction>]
                              [--fields <field> [<field> ...]]
ironic volume-connector-show [--fields <field> [<field> ...]] <uuid>
ironic volume-connector-update <uuid> <op> <path=value> [<path=value> ...]

ironic node-volume-connector-list [--detail] [--limit <limit>]
                                  [--marker <uuid>] [--sort-key <field>]
                                  [--sort-dir <direction>]
                                  [--fields <field> [<field> ...]]
                                  <node>
```

- A new `VolumeTargetManager` will be added to `ironicclient` to get/set target information for the node. Also the CLI will be modified as follows:

```
ironic volume-target-create --node <node> --volume_type <volume_type>
                            --volume_id <volume_id>
                            [--properties <key=value>]
                            [--boot_index <boot_index>]
                            [-e <key=value>] [-u <uuid>]
ironic volume-target-delete <uuid> [<uuid>]
ironic volume-target-list [--detail] [--volume_type <volume_type>]
                          [--volume_id <volume_id>]
                          [--boot_index <boot_index>] [--limit <limit>]
                          [--marker <uuid>] [--sort-key <field>]
                          [--sort-dir <direction>]
                          [--fields <field> [<field> ...]]
ironic volume-target-show [--fields <field> [<field> ...]] <uuid>
ironic volume-target-update <uuid> <op> <path=value> [<path=value> ...]

ironic node-volume-target-list [--detail] [--limit <limit>]
                               [--marker <uuid>] [--sort-key <field>]
                               [--sort-dir <direction>]
                               <node>
```

- New objects, `CreateBaremetalVolumeConnector`, `DeleteBaremetalVolumeConnector`, `ListBaremetalVolumeConnector`, `SetBaremetalVolumeConnector`, `ShowBaremetalVolumeConnector`, and `UnsetBaremetalVolumeConnector` will be added to `openstackclient` plugin to get/set connector information for the node. Also the CLI will be modified as follows:

```
openstack baremetal volume connector create [-h]
                                            [-f {json,shell,table,value,
↪yaml}]
                                            [-c COLUMN]
                                            [--max-width <integer>]
                                            [--noindent] [--prefix PREFIX]
                                            --node <node_uuid> --type
↪<type>
```

(continues on next page)

(continued from previous page)

```
                                          --connector_id <connector_id>
                                          [--extra <key=value>]
                                          [--uuid <uuid>]
openstack baremetal volume connector delete [-h] <connector> [<connector>]
openstack baremetal volume connector list [-h]
                                          [-f {json,shell,table,value,
↪yaml}]
                                          [-c COLUMN]
                                          [--max-width <integer>]
                                          [--noindent]
                                          [--quote {all,minimal,none,
↪nonnumeric}]
                                          [--limit <limit>]
                                          [--marker <uuid>]
                                          [--sort <key>[:<direction>]]
                                          [--long | fields <field [field]␣
↪...>]
openstack baremetal volume connector set [-h] [--node <node>]
                                         [--type <type>]
                                         [--connector_id <connector_id>]
                                         [--extra <key=value>] <connector>
openstack baremetal volume connector show [-h]
                                          [-f {json,shell,table,value,
↪yaml}]
                                          [-c COLUMN]
                                          [--max-width <integer>]
                                          [--noindent] [--prefix PREFIX]
                                          [--fields <field> [<field> ...]]
                                         <connector>
openstack baremetal volume connector unset [-h] [--extra <key>]
↪<connector>
```

- New objects, `CreateBaremetalVolumeTarget`, `DeleteBaremetalVolumeTarget`, `ListBaremetalVolumeTarget`, `SetBaremetalVolumeTarget`, `ShowBaremetalVolumeTarget`, and `UnsetBaremetalVolumeTarget` will be added to `openstackclient` plugin to get/set target information for the node. Also the CLI will be modified as follows:

```
openstack baremetal volume target create [-h]
                                         [-f {json,shell,table,value,yaml}
↪]
                                         [-c COLUMN] [--max-width
↪<integer>]
                                         [--noindent] [--prefix PREFIX]
                                         --node <node_uuid> --type <type>
                                         --volume_id <volume_id>
                                         [--properties <key=value>]
                                         [--boot_index <boot_index>]
                                         [--extra <key=value>]
                                         [--uuid <uuid>]
```

(continues on next page)

(continued from previous page)

```
openstack baremetal volume target delete [-h] <target> [<target>]
openstack baremetal volume target list [-h]
                                       [-f {json,shell,table,value,yaml}]
                                       [-c COLUMN] [--max-width <integer>]
                                       [--noindent]
                                       [--quote {all,minimal,none,
↪nonnumeric}]
                                       [--limit <limit>] [--marker <uuid>]
                                       [--sort <key>[:<direction>]]
                                       [--long | fields <field [field] ...
↪>]
openstack baremetal volume target set [-h] [--node <node>] [--type <type>]
                                      [--volume_id <volume_id>]
                                      [--properties <key=value>]
                                      [--boot_index <boot_index>]
                                      [--extra <key=value>] <target>
openstack baremetal volume target show [-h]
                                       [-f {json,shell,table,value,yaml}]
                                       [-c COLUMN] [--max-width <integer>]
                                       [--noindent] [--prefix PREFIX]
                                       [--fields <field> [<field> ...]]
                                       <target>
openstack baremetal volume target unset [-h]
                                        [--properties <key>]
                                        [--boot_index] [--extra <key>]
                                        <target>
```

RPC API impact

Four new rpcapi method `update_volume_connector`, `destroy_volume_connector`, `update_volume_target`, and `destroy_volume_target` will be added.

- `update_volume_connector`

  This method takes context and volume connector object as input and returns updated volume connector object.

- `destroy_volume_connector`

  This method takes context and volume connector object as input.

- `update_volume_target`

  This method takes context and volume target object as input and returns updated volume target object.

- `destroy_volume_target`

  This method takes context and volume target object as input.

Driver API impact

None.

Nova driver impact

When spawning a new instance, Nova Ironic virt driver queries Ironic (through API) to find out the volume connector information. It passes the volume connector information to Cinder which returns the target information. This is then passed down to Ironic. Detailed information about Nova Ironic driver can be found in the spec[5].

Ramdisk impact

None

Security impact

None.

Note

As for FC zoning, Cinder takes care of it[6].

Other end user impact

None.

Scalability impact

None.

Performance Impact

This may extend the time required for nova boot/delete, but its not a big impact and its important for enterprise users.

Other deployer impact

- If administrators want to provide boot from volume feature, they need to fill out following initiator information before activating the node.
  - iSCSI:
    - ip
    - iqn
    - mac

[5] https://review.opendev.org/#/c/211101/
[6] http://docs.openstack.org/mitaka/config-reference/block-storage/fc-zoning.html

> **Note**
>
> ip may be omitted when Neutron is used to manage the storage network.

  - FC:
    * wwnn
    * wwpn

  Administrators need to set the node.properties[capabilities] (iscsi_boot and/or fibre_channel_boot) true.

  Its better if inspection automatically collects and registers them. For example, in the case of a node with FC-HBA, inspection(in-band) can get wwnn and wwpn from sysfs like following:

```
# cat /sys/class/scsi_host/host*/device/fc_host/host*/node_name
# cat /sys/class/scsi_host/host*/device/fc_host/host*/port_name
```

- If users want to boot a node from volume in Ironic standalone mode, they need additional tooling to leverage this functionality. For example, that tool needs to do something like:
  - Get initiator information from Ironic
  - Call the storage management tool with initiator information to create a new volume (maybe from template) and attach it to the initiator
  - Get target information from storage management tool
  - Put target information into Ironic

Developer impact

Driver developers can consume the information mentioned above to write boot from volume support in their driver. The details about reference driver and driver interface specs are described in [4].

Implementation

Assignee(s)

Primary assignee:
satoru-moriya-br

Other contributors:
rameshg87

Work Items

- Create new table named volume_connectors and volume_targets
- Create new DB API methods
- Create new Object named VolumeConnector and VolumeTarget
- Create new RPC API methods
- Create new REST API endpoints

- Document the changes
- Enhance inspector to register connector information if available
- Enhance Client(CLI) to get/set connector and target information
- Enhance Nova-Ironic driver to support boot from volume with these APIs

Dependencies

None

Testing

- Unit tests will be added/updated to cover the changes.
- Tempest tests will be added to Ironic to ensure that the following newly added API endpoints work correctly.

Upgrades and Backwards Compatibility

Add a migration script for database.

Documentation Impact

Documentations such as Installation guide and api-ref will be updated to explain the newly added fields and end points.

- Installation guide:

  http://docs.openstack.org/developer/ironic/deploy/install-guide.html

- api-ref documentation:

  http://developer.openstack.org/api-ref/baremetal/index.html

References

5.15.207 Lenovo XClarity Driver

https://bugs.launchpad.net/ironic/+bug/1702508

This specification proposes to add new interfaces that provide Ironic support to Lenovo XClarity managed servers. Servers that are supported by XClarity today are provided in the references at the end of this specification.

Problem description

Lenovo servers use an IMM GUI which brings forth a unique way of managing the nodes. The nodes need to be created, managed and operated by the XClarity Administrator GUI tool and are identified by a unique host id.

In addition to managing the nodes using IPMI protocol, this specification proposes to add hardware types and interfaces to manage Lenovo servers using XClarity Administrator RESTful API. The REST API documentation for XClarity is provided in the references at the end of the document. Benefits of using XClarity over plain IPMI are: better user management, hardware monitoring and hardware management.

Proposed change

New power and management interfaces will be added as part of this change.

The interfaces use RESTful API to communicate with XClarity Administrator. The interfaces used are:

- XClarity.XClarityPower for Power operations
- XClarity.XClarityManagement for Management operations

The XClarity Administrator RESTful API provides various operations, like controlling the power of nodes, retrieving firmware version and Feature on Demand(FoD) enablement information, etc.

The multiple interfaces embed the client side code for communicating with XClarity API via HTTP/HTTPS protocol, as a simple wrapper class. For all the interfaces, an xclarity-client will to be imported. The xclarity-client is available at https://pypi.python.org/simple/xclarity-client/.

- Power:

  This feature allows the user to turn the node on/off or reboot by using the power interface which will in turn call XClaritys REST API.

- Management:

  This feature allows the user to get and set the primary boot device of the Lenovo servers, and to get the supported boot devices.

Alternatives

Use of the generic IPMI interfaces and pre-existing deploy interfaces is an alternative especially in mixed configurations.

Data model impact

None

RPC API impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

Driver API impact

None

Nova driver impact

None

Security impact

Use of HTTPS for this interface, and verifying certificates is more secure than IPMI-based interfaces.

Lenovo XClarity Administrator uses certificates to establish secure, trusted communications between XClarity and its managed devices. By default, the managed devices use XClarity-generated certificates. The user can choose to let Lenovo XClarity Administrator manage certificates, or customize and replace the server certificates. XClarity provides options for customizing certificates depending on the users requirements.

Other end user impact

None

Scalability impact

There are some locking considerations when we are dealing with scale. All locking is within a primary job/task framework which are profile activation, firmware update, and OS deploy. A client will initiate a job on an XClarity API endpoint, at which time it is locked from other jobs or client actions until the job completes.

There is an additional mechanism used by Firmware Updates to prevent inventory from refreshing from events, so commands are not sent to the IMM during the update. A second level of locking is when a client attempts power actions, and during the power request the endpoint is locked. This is generally very quick though.

Performance Impact

Lenovo XClarity Administrator supports the management of up to 20 chassis with compute nodes and a similar number of rack servers. An operator with a large number of systems being managed by XClarity should expect reduced system performance. Performance considerations have been provided in references at the end of this specification.

Ramdisk impact

None

Other deployer impact

The following driver_info fields are required while enrolling node into Ironic:

- xclarity_address: XClarity Administrator IP-Address
- xclarity_username: XClarity Administrator username
- xclarity_password: XClarity Administrator password

- xclarity_hostid: The host ID that is allocated by XClarity Administrator for each managed host.
- xclarity_port(optional): The port used for establishing xClarity Administrator connection. Default is 443.

Developer impact

None

Implementation

Assignee(s)

Primary assignee:
Hu Bian (hubian1@lenovo.com)

Other contributors:
Jia Pei (jiapei2@lenovo.com) Haijun Mao (maohj2@lenovo.com) Finix Lei (leilei4@lenovo.com) Rushil Chugh (crushil)

Work Items

- Add new XClarity hardware type, and adding new interfaces for Power and Management.
- Writing appropriate unit tests to provide test coverage for XClarity driver.
- Writing configuration documents.
- Third party CI-setup.

Dependencies

None

Testing

- Unit tests will be implemented for new XClarity driver.
- Third party Continuous integration (CI) support will be added for Lenovo servers.

Upgrades and Backwards Compatibility

None

Documentation Impact

- Updating Ironic documentation section *Enabling Drivers* with XClarity related instructions.

References

- XClarity Restful API introduction
- XClarity Supported servers
- XClarity Performance considerations <http://flexsystem.lenovofiles.com/help/index.jsp?topic=%2Fcom.lenovo.lxca.doc%2Fplan_performconsiderations.html>_

CHAPTER

SIX

RETIRED

These specifications are ideas and features which are no longer applicable. They were either not implemented, no longer exist, and have generally been superseded either through the evolution of the projects capabilities or the state in the marketplace. Each document listed below has been updated to indicate the context as to why it was moved to this list.

6.1 Discover node properties and capabilities for ucs drivers

https://bugs.launchpad.net/ironic/+bug/1526359

This proposal adds the ability to inspect/update hardware properties and auto-create ports for Cisco UCS B/C/M-servers managed by Cisco UCS Manager. It uses out-of-band H/W inspection utility provided by UcsSdk.

Note

This specification has been retired as the vendor specific UCS driers are no longer available in Ironic. This was a result of the Open Source UcsSdk library no longer being maintained.

6.1.1 Problem description

Node inspection automatically collects node properties. These properties are required for scheduling or for deploy (ports creation). Today UCS drivers doesnt support node inspection. Enhance UCS drivers to support node inspection.

6.1.2 Proposed change

This spec proposes change to enhance UCS drivers to support node inspection that discovers node properties and capabilities of Cisco UCS B/C/M-series servers managed by Cisco UCSM. This is done by using out-of-band H/W inspection interface provided by UcsSdk Python library.

The following mandatory properties will be discovered and updated in node.properties as discussed in http://specs.openstack.org/openstack/ironic-specs/specs/kilo/ironic-node-properties-discovery.html

- memory size
- CPUs
- CPU architecture
- NIC(s) MAC address
- disks

The following additional properties are of interest to UCS drivers and will be discovered and updated to node.properties as capabilities:

- UCS Host Firmware pack

  capability name : ucs_host_firmware_package possible values : it can vary hardware to hardware.

- Server Name/Model

  capability name : server_model possible values : it can vary hardware to hardware.

- RAID level

  capability name : max_raid_level possible values : 0,1,5,6,10

- secure boot capability

  capability name : secure_boot possible values : true, false

- PCI (GPU) devices

  capability name : pci_gpu_devices possible values : count of such devices.

- SR-IOV capabilities

  capability name : sr_iov_devices possible values : count of such devices.

- NIC Capacity

  capability name : nic_capacity possible values : value with unit.

- TPM Support

  capability name : trusted_boot possible values : true, false

- Multi LUN support

  capability name : multi_lun possible values : true, false

- CDN (Consistent Device Name) Support

  capability name : cdn possible values : true, false

- VXLAN Capability

  capability name : vxlan possible values : true, false

- NV GRE Capability

  capability name : nv_gre_devices possible values : count of such devices

- NET FLOW Capability

  capability name : supports_net_flow possible values : true, false

- FlexFlash Capability

  capability name : flex_flash possible values : true, false

- UCS service-profile template name

  capability name : ucs_sp_template possible values : service profile template name

The properties which are already set will be overridden at reinvocation of inspect_hardware() except for NICs. If a port already exists, it will not create a new port for that MAC address. It will take care of adding as well as deleting of the ports for NIC changes [2]. Not all the capabilities are applicable to all Cisco UCS B/C/M-series server models. If a property is not applicable to the hardware, the property

will not be added/updated in node.properties as capabilities. Inspection fetches only those capabilities applicable to the specific server model.

Inspection returns failure in the following cases:

- Failed to get basic properties.
- Failed to get capabilities, due to service-profile configuration errors.
- Communication errors with UCS Manager.

UCS specific module changes:

- Implement the InspectInterface method inspect_hardware().

Alternatives

These properties can be discovered manually outside the ironic and node.properties updated accordingly with the discovered properties.

Data model impact

None.

State Machine Impact

None

REST API impact

None.

Client (CLI) impact

None

RPC API impact

None.

Driver API impact

None.

Nova driver impact

None.

Ramdisk impact

N/A

Security impact

None.

Other end user impact

None.

Scalability impact

None.

Performance Impact

None.

Other deployer impact

None.

Developer impact

None.

6.1.3 Implementation

Assignee(s)

Primary assignee:
saripurigopi

Work Items

- Implementation of the InspectInterface class and its methods inspect_hardware(), validate() and get_properties().

6.1.4 Dependencies

- This feature is targeted for Cisco UCS B/C/M-series servers managed by UCS Manager 2.2(4b) or above. All the capabilities listed might not be available with older versions of UCS Manager (like 2.2(3b)).
- Depends on UcsSdk library.

6.1.5 Testing

Unit tests will be added conforming to ironic testing requirements, mocking UcsSdk. It will get tested on real hardware by UCS team with the available hardware models to the team.

6.1.6 Upgrades and Backwards Compatibility

No impact.

6.1.7 Documentation Impact

Hardware Inspection section will be added and updated accordingly in doc/source/drivers/ucs.rst.

6.1.8 References

1. UcsSdk library: * https://github.com/CiscoUCS/UcsSdk * https://pypi.org/project/UcsSdk

2. Introspect spec: * https://github.com/openstack/ironic-specs/blob/master/specs/kilo/ironic-node-properties-discovery.rst

6.2 Out-of-band RAID configuration using Cisco UCS drivers

https://bugs.launchpad.net/ironic/+bug/1526362

This blueprint proposes to implement out-of-band RAID configuration interface for Cisco UCS drivers. This implementation supports configuration of Cisco UCS Manager (UCSM) managed B/C/M-series servers.

Note

This specification has been retired as the vendor specific UCS driers are no longer available in Ironic. This was a result of the Open Source UcsSdk library no longer being maintained. Users may wish to explore use of the `redfish` driver, but it is unknown to the community if the UCS Redfish support has been extended to RAID support.

6.2.1 Problem description

Currently pxe_ucs and agent_ucs drivers dont support RAID configuration on UCSM managed servers.

6.2.2 Proposed change

It proposes implementing the RAID interface as described by the parent spec[1] for UCS drivers.

List of changes required:

- ucs.raid.RAIDInterface for RAID configuration operations

  The following methods will be implemented:

  - validate
  - create_configuration
  - delete_configuration

  validate() method validates the required UCS parameters for OOB RAID configuration. Also calls validate() of the super class to validate json schema. create_configuration and delete_configuration operations are implemented as asynchronous RAID configuration deployment operations by UCS drivers. UcsSdk/UCS-API asynchronously deploys the RAID configuration on the target node. UCS driver(s) sends the RAID configuration simultaneously to the target node when the operation is invoked, but UCS Manager would not deploy the configuration simultaneously on the target node. UCS Manager accepts the RAID configuration and deploys it as part of UCS Manager FSM deploy state. Hence there will be delay between, when the operation is invoked and when the RAID configuration is deployed. To know the deploy status, we need to query the FSM status using UcsSdk API. These methods return states.CLEANWAIT. New driver

[1] New driver interface for RAID configuration: https://review.opendev.org/173214

periodic task will be added to fetch the UCSM FSM status of these operations. This periodic task is enabled only if pxe_ucs, agent_ucs drivers are enabled in the conductor.

- RAID management changes:

  Controlling the RAID configuration is creating storage-profile ManagedObject and associating with the Server object in UCS Manager 2.4. Earlier version of UCSM requires to configure LocalDiskConfigPolicy and associate with respective service-profile ManagedObject. The service-profile information is captured as part of node driver_info properties. UcsSdk provides RAIDHelper interface, which actually creates the required policies specified above. UCS driver uses this interface and makes appropriate calls to create_config and delete_config operations.

Alternatives

Operator can change the RAID configuration manually whenever required after putting the node to MANAGEABLE state. But this has to be done for each node.

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

N/A

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

None

Developer impact

None

6.2.3 Implementation

Assignee(s)

Primary assignee: saripurigopi

Work Items

- Add UcsRAIDManagement inheriting from base.RAIDInterface for ucs drivers.
- Writing and unit-test cases for RAID interface of ucs drivers.
- Writing configuration documents.

6.2.4 Dependencies

- UcsSdk to support RAID configuration utility

6.2.5 Testing

Unit-tests will be implemented for RAID interface of ucs driver.

6.2.6 Upgrades and Backwards Compatibility

Adding RAID interface support for ucs drivers will not break any compatibility with either REST API or RPC APIs.

6.2.7 Documentation Impact

- Writing configuration documents.

6.2.8 References

6.3 iPXE Dynamic Configuration

https://bugs.launchpad.net/ironic/+bug/1526275

This adds support for dynamically generating iPXE configuration files when booting a node.

6.3.1 Problem description

The current iPXE support depends on configuration files to be cached on the disk. This creates a dependency between a given `ironic-conductor` and a given node (even without a conductor lock on a node) because that `ironic-conductor` is the only one able to boot that node. This also makes take-over more complicated because the new `ironic-conductor` will need to regenerate the iPXE configuration files for the new nodes its now managing and update the DHCP server accordingly.

6.3.2 Proposed change

The proposed implementation consists of creating a new `Driver Vendor Passthru` method called `ipxe_config` that will dynamically generate the iPXE configuration files for a given node UUID or mac address depending on the nodes provision state.

When Neutron is used with iPXE enabled, it will configure the DHCP server to make a request to the `Driver Vendor Passthru` endpoint using the nodes UUID when booting a node, e.g:

```
http://<Ironic API Address>:6385/v1/drivers/<driver_name>/vendor_passthru/
↪ipxe_config?node_uuid=<node UUID>
```

Ironic will then check the `provision_state` of the node and generate the iPXE configuration file for that state. Say, the node `provision_state` is DEPLOYING, we then will return an iPXE configuration to boot the deploy ramdisk and kernel. If the node `provision_state` is ACTIVE, we then return an iPXE configuration to boot from the image ramdisk and kernel (If local boot and/or full disk image is not specified). For an unknown `provision_state` we just return an iPXE configuration file that prints out an error explaining the problem on the nodes console log and a warning message in the Ironic log.

If an operator wants to have an external DHCP server (standalone version) but still benefit from dynamically generated iPXE script files (instead of using static files) it will be possible by making the same `Driver Vendor Passthru` endpoint to support passing the MAC address of one of the nodes port as parameter, e.g:

```
http://<Ironic API Address>:6385/v1/drivers/<driver_name>/vendor_passthru/
↪ipxe_config?port_address=<port address>
```

When scripting iPXE allows expanding variables so that an operator can create a single iPXE script pointing to the Ironic API (and expanding the `${mac}` variable) when configuring their external DHCP server allowing them to have dynamically generated iPXE configuration for their environment even when Neutron is not used.

This work can get even more powerful when the images are set to boot from `http`[1], as then the iPXE drive wont need to save any state on the disk. As a future work, it would be also possible to add support for creating a `Swift Temporary URL` when booting images being served by `Glance` with a `Swift` storage backend.

Alternatives

Continue doing what we are doing, generate the configuration files and saving it to the disk.

[1] http://specs.openstack.org/openstack/ironic-specs/specs/kilo/non-glance-image-refs.html

Data model impact

None

State Machine Impact

None

REST API impact

A new `Driver Vendor Passthru` method called `ipxe_config` that supports GET HTTP.

Client (CLI) impact

None

RPC API impact

Currently the RPC method for `vendor_passthru` and `driver_vendor_passthru` returns a tuple with the return value and a boolean indicating if the method is asynchronous. We will need another flag to indicate if the value should be returned as a static file that will be served by the Ironic API instead of a response body message.

Driver API impact

None

Nova driver impact

None

Ramdisk impact

N/A

Security impact

The new `Vendor Passthru` method endpoint needs to be part of the public API, so that iPXE can get the configuration file from without authentication. This is the same as the methods `heartbeat` or `lookup` for the agent driver[2].

Other end user impact

None

Scalability impact

A stateless driver can scale better since it wont depend on any information to be saved on the local conductor.

[2] https://github.com/openstack/ironic/blob/master/ironic/api/config.py

Performance Impact

None

Other deployer impact

None

Developer impact

None

6.3.3 Implementation

Assignee(s)

Primary assignee:
lucasagomes <lucasagomes@gmail.com>

Other contributors:

Work Items

- Create the new `ipxe_config` method for the PXEVendorPassthru interface.
- Change the PXE configuration options passed to the DHCP server to point to the `v1/drivers/<driver name>/vendor_passthru/ipxe_config?node_uuid=<node UUID>` endpoint in the Ironic API instead of pointing to the URL to download the boot.ipxe script (the script wont be need anymore and will be deleted).
- Extend the `vendor_passthru` and `driver_vendor_passthru` RPC methods to return a flag indicating whether the return value should be attached to the response object as a file or returned as a response message.
- Update the methods `prepare_ramdisk` and `clean_up_ramdisk` from the **IPXEBoot** interface to not attempt to create or delete the iPXE configuration files.

6.3.4 Dependencies

- New boot interface: This spec is refactoring the boot logic out of the current Ironic `deploy` drivers into a new boot interface.

6.3.5 Testing

Unittests will be added.

6.3.6 Upgrades and Backwards Compatibility

None

6.3.7 Documentation Impact

The iPXE documentation will be updated to reflect the changes made by this spec.

6.3.8 References

6.4 OpenBMC Driver

https://bugs.launchpad.net/ironic/+bug/1634635

This proposal covers the addition of power and management interfaces for OpenBMC and the addition of driver to the IPA and PXE driver classes.

Note

This specification is no longer viewed as applicable as OpenBMC has support for the Redfish API interface, and as such this is considered a superior standards based interface for use. When this specification was proposed, These the Redfish API had not yet been implemented in OpenBMC.

6.4.1 Problem description

Currently IPMI is not fully supported by OpenBMC. Instead, OpenBMCs power is controlled through a RESTful interface. As well, the boot device can be retrieved and set through this RESTful interface. While IPMI may be supported by OpenBMC in the future, the recommended way to interact with it will continue to be its REST API.

Servers running OpenBMC will require a new interface implementation to control its power, another to set the boot device, and a hardware type that use has these interfaces supported.

6.4.2 Proposed change

The addition of an OpenBMCPower() module conforming to base.PowerInterface. Login credentials will be specified as openbmc_address, openbmc_username, and openbmc_password in the driver_info property of the node.

The addition of an OpenBMCManagement() module conforming to base.ManagementInterface which also uses the openbmc_address, openbmc_username, and openbmc_password as login credentials.

The module will login to the BMC, issue the proper command, then log out of the BMC.

A hardware type, OpenBMCHardware, will be added. This hardware type will have OpenBMCPower in its supported_power_interfaces list. This hardware type will also have OpenBMCManagement in its supported_management_interfaces list.

Alternatives

Wait for IPMI functionality to be fully supported by OpenBMC. This would allow the pxe_ipmitool and agent_ipmitool drivers to work.

The disadvantage here is that it is not the recommended method of interaction with the BMC. As well, it is unknown when IPMI will be fully supported.

Data model impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

ironic CLI

None

openstack baremetal CLI

None

RPC API impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

None

Security impact

None

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

None

Developer impact

None

6.4.3 Implementation

Assignee(s)

Primary assignee:
Michael Turek (mjturek)

Other contributors:
Mark Hamzy (mark-hamzy)

Work Items

- Implement a new power interface, OpenBMCPower, conforming to base.PowerInterface.
- Implement a new management interface, OpenBMCManagement, conforming to base.ManagementInterface
- Add hardware type, OpenBMCHardware, that has these interfaces as supported.
- Add documentation detailing usage of interfaces and driver.

6.4.4 Dependencies

This feature will only be usable by target hardware that runs OpenBMC.

6.4.5 Testing

The feature will be tested using the KVM on POWER OpenStack CI environment.

The job will run the ironic tempest tests, but no new integration tests will be added. The job will test against real hardware initially.

Unit tests will be added as well.

6.4.6 Upgrades and Backwards Compatibility

None

6.4.7 Documentation Impact

Documentation will be added describing the new interfaces and how to use them.

6.4.8 References

- OpenPOWER
- OpenBMC REST API Examples
- KVM on POWER OpenStack CI

6.5 Petitboot boot driver

https://bugs.launchpad.net/ironic/+bug/1526265

This adds petitboot boot driver for OpenPOWER servers. The OpenPOWER Foundation is a collaboration around Power Architecture products initiated by IBM.

Note

This specification has been retired as the IBM firmware maintainers added support for the classical PXELinux file format. As such, this specification is no longer needed.

6.5.1 Problem description

OpenPOWER servers use FSP (flexible service processor) to initialize the system chip and monitor the other hardware components. FSP can be set to OPAL (Open Power Abstract Layer) mode to handle the IPMI operation. The firmware on OpenPOWER servers use petitboot as a platform independent bootloader. Petitboot can load kernel, initrd and device tree files from any Linux mountable filesystem, plus can load files from the network using the FTP, SFTP, TFTP, NFS, HTTP and HTTPS protocols.

Petitboot is very similar to the standard pxe behaviour, but a little difference, like:

- Petitboot will be loaded every time no matter what the boot device is.
- Petitboot will check the system configuration like boot device in the firmware and will not download any netboot loader, as itself can be worked as a netboot loader.
- If boot device is hard disk, petitboot will scan boot or prep partition to load the boot option from the local boot loader, for example grub2, then let grub2 load the system.
- If boot device is network, petitboot will look for PXE configuration file (209) option in the dhcp response. If no explicit configuration file is given, then petitboot will attempt `pxelinux-style` configuration auto-discovery, using the machines MAC address, the IP of the DHCP lease, and fall back to a file named default.
- Petitboot requests images from network according to the path information in the configuration file, then use kexec to load the system. So both the deploy system and the instance system should support boot from kexec when the boot device is network.
- The format of petitboot configuration file for netboot is different from the `pxelinux.cfg` and more kernel command parameters about network should be passed.

6.5.2 Proposed change

- Add `iscsi_opc` and `agent_opc` drivers. `opc` means OpenPOWER controller which use petitboot driver as the boot driver and ipminative driver as the hardware control driver. For example, the iscsi_opc driver will look like this

```
class OpenPOWERIscsiAndIPMINativeDriver(base.BaseDriver):
"""Petitboot + Iscsi + IPMINative driver"""
  def __init__(self):
    self.power = ipminative.NativeIPMIPower()
    self.console = ipminative.NativeIPMIShellinaboxConsole()
    self.boot = petitboot.PetitbootBoot()
    self.deploy = iscsi_deploy.ISCSIDeploy()
```

(continues on next page)

(continued from previous page)

```
    self.management = ipminative.NativeIPMIManagement()
    self.vendor = iscsi_deploy.VendorPassthru()
    self.inspect = inspector.Inspector.create_if_enabled(
        'OpenPOWERIscsiAndIPMINativeDriver')
```

- Add configuration option for petitboot driver in ironic.conf

```
[petitboot]
config_template: template path for petitboot configuration.
protocol: string value for the transfer protocol, only support http and
          tftp in this spec, default http.
```

- Add Petitboot driver inherits base.BootInterface and implements the following functions.
  - `validate()` - Check boot option and image type. In this spec, OpenPOWER machine only support local boot with whole disk image and netboot with partition images.
  - `get_properties` - In this spec, return common properties which are as same as the properties of pxe driver.
  - `prepare_ramdisk()` - Petitboot driver will update the dhcp option, build the configuration file for the petitboot netboot loader, then set the boot device to network. This procedure is similar to the pxe driver, but a little difference.
    * Petitboot do not need NBP file, so petitboot driver will not pass `bootfile-name` (67) option to dhcp. Petitboot driver will use PXE configuration file (209) option and PXE path prefix (210) option to locate the configuration file. Petitboot will concatenate these values to generate the file transfer request. The value of 209 option is the relative file path like `<node_uuid>/config`. The value of 210 option depends on the `protocol` configuration in the petitboot section.
      · If it is tftp, the value of 210 option is `CONF.deploy.tftp_root`
      · If it is http, the value of 210 option is `CONF.deploy.http_url`.
    * Petitboot driver will manage this configuration file which contains the path information for the kernel and ramdisk. The format of the configuration file is like

```
default deploy

label deploy
kernel ``{{ [petitboot_url]/<node_uuid>/deploy_kernel }}``
initrd ``{{ [petitboot_url]/<node_uuid>/deploy_ramdisk }}``
append ``{{ parameter }}``
```

  - `clean_up_ramdisk()` - Cleans up the environment that was setup for booting the deploy system. It unlinks the deploy kernel/ramdisk.
  - `prepare_instance()` - This method is very similar to the pxe driver.
    * If boot option is local, just clean up the petitboot configuration file.
    * If boot option is network, update the petitboot configuration file to boot the instance image.

- `clean_up_instance()` - Cleans up the environment that was setup for booting the instance. It unlinks the instance kernel/ramdisk and removes the petitboot config.

Alternatives

None

Data model impact

None

RPC API impact

None

State Machine Impact

None

REST API impact

None

Client (CLI) impact

None

Driver API impact

None

Nova driver impact

None

Ramdisk impact

A ramdisk capable of running on PPC64 hardware will need to be built, however, this may be done downstream.

Support should be added to ramdisk build tooling, such as `disk-image-builder` and `coreos-image-builder`, to build such ramdisks.

Security impact

None

Other end user impact

To use petitboot driver for the OpenPOWER servers, the `cpu_arch` in driver properties should be ppc64le or ppc64 which depends on the cpu architecture of instance image. OpenPOWER servers can switch to the appropriate endian format according to the endian format of kernel image. Both the deploy kernel and the instance kernel should support boot from kexec when local boot is not enabled.

Scalability impact

None

Performance Impact

None

Other deployer impact

New config options

```
[petitboot]
config_template: template path for petitboot configuration.
protocol: string value for the transfer protocol, only support http
          and tftp in this spec, default http.
```

Developer impact

None

6.5.3 Implementation

Assignee(s)

Primary assignee:
chenglch <chenglch@cn.ibm.com>

Other contributors:
baiyuan <bybai@cn.ibm.com>

Work Items

- Implement petitboot boot driver.
- Add `iscsi_opc` and `agent_opc` drivers to manage the OpenPOWER servers.
- Write unit-test cases.
- Write configuration documents.

6.5.4 Dependencies

None

6.5.5 Testing

- Unit Tests
- Third-party CI Tests: We have plan to build 3rd-party CI for this driver, but do not have sufficient hardware available at this time.

6.5.6 Upgrades and Backwards Compatibility

This driver will not break any compatibility with either on REST API or RPC APIs.

6.5.7 Documentation Impact

Writing documents to instruct operators how to use Ironic with petitboot driver.

6.5.8 References

- OpenPOWER
- petitboot
- Netbooting with petitboot

6.6 Snapshot support

https://storyboard.openstack.org/#!/story/2008033

6.6.1 Problem description

Snapshot is not a new thing, it was available for virtual machines for a long time. Snapshot is useful for instance backup, image reusing, etc, but there is no such support for bare metals as its more complex than passing the request to libvirt.

Bare metal snapshot may not match with virtual machines in speed and efficiency, but it could address following requirements:

As an operator, I want to be able to back up a bare metal instance periodically and when there is hardware failure, the same image can be applied to another machine.

As an operator, I want to be able to build a master image from a post customized instance, capture the system into an image and apply to other similar machines.

6.6.2 Proposed change

The proposal is to implement a similar process like deployment, when a node is requested to do a snapshot, ironic prepares boot configuration then triggers a netboot to the node, IPA lookups and ironic instructs IPA to streaming the data of root disk to a remote storage.

The remote storage could be the host where the conductor is running if proper service like NFS, HTTP, etc is configured. Or another host as long as both conductor and IPA are able to access. The remote storage will act as an intermediate storage, when the streaming is done, conductor will upload the image to the image service and remove the image in the intermediate storage.

The initial proposal is to utlilize HTTP WebDAV as remote storage, NFS and other storage can be extended when this feature is implemented. When the direct deploy interface is configured to download images from HTTP service, it can be updated with WebDAV support.

Alternatives

None

Data model impact

A new hardware interface `snapshot_interface` will be added to support different snapshot implementations.

State Machine Impact

Add following states to the state machine:

- snapshot wait
- snapshotting
- snapshot failed

With following transitions:

- active -> snapshotting (on snapshot) Prepare boot and network
- snapshotting -> snapshot failed (on fail) On error
- snapshotting -> snapshot wait (on wait) Wait for ramdisk alive
- snapshot wait -> snapshotting (on resume) Image streaming and uploading
- snapshot wait -> snapshot failed (on fail) On timeout
- snapshot wait -> snapshot failed (on abort) On abort request
- snapshotting -> active (on done) Snapshot complete
- snapshot failed -> active (on abort) Abort a failed snapshot
- snapshot failed -> snapshotting (on snapshot) Retrigger a failed snapshot
- snapshot wait -> deleting (on deleted) Abort snapshot and undeploy
- snapshot failed -> deleting (on deleted) Abort and undeploy

REST API impact

The existing provision endpoint is used, a new verb `snapshot` will be guarded by microversions.

- PUT /v1/nodes/{node_ident}/states/provision

  When the requested version is permitted, and node is in the active state, start to do a snapshot. An `image_ref` argument is required, which refers to a location where the snapshot image should be stored.

  If the `image_ref` is an UUID, it refers to the UUID of an image in the Image service, the image should be precreated before snapshot request and acts as a holder for receiving image data. When integrated with Compute service, the `image_id` is an argument passed to the driver interface `snapshot()`.

  Example request:

```
 {
   "target": "snapshot",
   "image_ref": "66498c26-a9b5-496c-97a8-5bc08f256155"
 }

The ``image_ref`` could be an URL in other forms for standalone usage,
but not covered in this spec.
```

Other provision state transitions will use existing verbs.

Client (CLI) impact

openstack baremetal CLI

The impact to CLI should be trivial, will have a new command:

- `openstack baremetal node snapshot <node> ...`

openstacksdk

openstacksdk will be enhanced to know the snapshot API version.

RPC API impact

Will introduce a new rpc interface `do_node_snapshot`.

Driver API impact

Adds an optional interface `snapshot_interface`, initially implements `fake`, `no-snapshot`, and `agent`.

For the `agent` implementation, conductor host needs a WebDAV service to receive image data from the IPA. When the first heartbeat is received, ironic conductor sends command `snapshot.stream_image` with an URL to the IPA to start the snapshotting. IPA find the root device and use proper methods to dump disk data to the remote storage. The criterial for choosing the root device is same with deployment, on the time of this writing, the root device is the first available block device not less than 4GiB, or the first matched device if the root device hint is specified.

There are two methods to utilize the WebDAV service:

- The WebDAV directory is mounted to the ramdisk, IPA uses `qemu-img` for snapshot, empty blocks will be bypassed so it provides better performance.
- IPA uses `dd` to dump the disk data to the WebDAV URL, when finished, ironic conductor needs to convert the raw image into proper format.

As the root device could be quite large and even larger than the available space of conductor host, `dd` is not considered practical in the spec.

After the disk data is successfully retrieved, the conductor is responsible to upload the image to the Image Service using the specified `image_id` and remove the intermediate image.

Nova driver impact

Nova driver will need to implement the driver interface `snapshot` to integrate with ironic. But before the integration the feature can be consumed by ironic as long as the Image Service is available.

Ramdisk impact

For the `agent` implementation, a `snapshot` extension will be added to the ironic-python-agent, when the WebDAV directory could be mounted, `qemu-img` is used for streaming disk data to the URL.

The snapshot extension will be defined as:

Example arguments for an HTTP WebDAV connection:

- url: http://10.10.1.1:8080/snapshots
- image_name: a830ebe1-67d4-448f-aa10-5bb33f3f3c02-snapshot.qcow2

IPA will generate an URL to mount:

```
mount -t davfs http://10.10.1.1:8080/snapshots /tmp/tmp5fjchif0
```

Note

NFS connection is not covered but can be implemented after this feature is implemented. It will look like:

```
mount -t nfs 10.10.1.1:/var/lib/ironic/snapshots /tmp/tmp5fjchif0
```

Security impact

The transfer of instance image from the target bare metal to image service could have security risk since the data could be tampered or retrieved during this process.

Other end user impact

None

Scalability impact

None

Performance Impact

None

Other deployer impact

None

Developer impact

None

6.6.3 Implementation

Assignee(s)

Primary assignee:
<kaifeng, kaifeng.w@gmail.com>

Other contributors:
<TheJulia, juliaashleykreger@gmail.com>

Work Items

- Equiping IPA with the ability of streaming disk data
- Spawning states and transitions to Ironic
- DB, RPC and API change
- Make Client/SDK aware of the feature
- Plug in with nova

6.6.4 Dependencies

None

6.6.5 Testing

Will be covered by unit tests and tempest.

6.6.6 Upgrades and Backwards Compatibility

Should be backwards compatible.

6.6.7 Documentation Impact

Documentation will be updated on this feature.

6.6.8 References

- https://etherpad.opendev.org/p/PVG-ironic-snapshot-support

6.7 VNC Graphical console

https://bugs.launchpad.net/ironic/+bug/1567629

In addition to a serial console, allow ironic nodes to be accessed through a graphical console. This proposal presents the work required to create a new driver interface for accessing graphical console of a node.

Note

This specification has been retired. This was done based upon community consensus on several aspects of the design. Primarily, in that the existing console interface can serve both serial consoles and graphical consoles without the overhead and operator confusion induced through yet another different interface to set. From a secondary aspect, serial console use and interaction has fallen from favor since this feature was designed, specifically in part due to move to UEFI firmware and the overall modeling of serial ports in modern hardware, as a primary output driven by CPU interrupt triggering, which can have detrimental performance impacts on running workloads.

6.7.1 Problem description

End users often have to troubleshoot their instances because they might have broken their boot configuration or locked themselves out with a firewall. Keyboard-Video-Mouse (KVM) access is often required for troubleshooting these types of issues as serial access is not always available or correctly configured. Also, KVM provides a better user experience as compared to serial console.

Currently, ironic does not expose a nodes KVM capabilities. As such, admin users and deployers have to find alternatives to provide KVM access to their users. Also, Horizons VNC console is not supported for the ironic nodes provisioned by Nova.

6.7.2 Proposed change

- A new interface `GraphicalConsoleInterface` will be added. This interface will essentially have the same class API as current `ConsoleInterface` interface (with `start_console`, `stop_console` and `get_console` methods), but it will be possible to enable/disable/configure it independently from serial console access. As with other ironic driver interfaces and hardware types, operators are free to choose which implementation of a graphical console access to use by setting it to the one enabled and supported by corresponding hardware type implementations. The new interface will have following methods:

```
class GraphicalConsoleInterface(BaseInterface):
    """Interface for graphical console-related actions."""
    interface_type = "graphical_console"

    @abc.abstractmethod
    def start_console(self, task):
        """Start a remote graphical console for the task's node.

        This method should not raise an exception if console already␣
↪started.

        :param task: a TaskManager instance containing the node to act on.
        """

    @abc.abstractmethod
    def stop_console(self, task):
        """Stop the remote graphical console session for the task's node.

        :param task: a TaskManager instance containing the node to act on.
        """

    @abc.abstractmethod
    def get_console(self, task):
        """Get connection information about the graphical console.

        This method should return the necessary information for the
        client to access the graphical console.

        :param task: a TaskManager instance containing the node to act on.
        :returns: the graphical console connection information.
        """
```

- The following new hardware interface implementations of `GraphicalConsoleInterface` will be created.
  - `ipmitool-vnc` - For accessing graphical console using VNC.
  - `no-graphical-console` - For no graphical console.
  - `fake` - For accessing fake graphical console used for testing.
- New config options will be introduced for this interface which are as follows:
  - `[DEFAULT]enabled_graphical_console_interfaces` - This config option represents

the list of enabled graphical console interfaces in ironic. The default value is `['no-graphical-console']`.

  - `[DEFAULT]default_graphical_console_interface` - This config option represents the default graphical console interface to be used with various drivers. The default value will be `no-graphical-console`.

- Two new fields will be added to the Node object:
  - `graphical_console_interface` - This field represents the supported graphical console interface for the node.
  - `graphical_console_enabled` - This field will a Boolean value that will represent the state of console. It will be set to True via request to start the graphical console.
- While a node unprovisioning, Ironic will stop all the graphical connections to the node.

Alternatives

- Accept this limitation and only offer a serial console.
- Use out-of-band KVM access provided by administrator without Ironic support.
- Generalize and formalize concept of a `console` interface, and allow to have arbitrary number of console interfaces (from those declared as supported by a hardware type) to be active and enabled for a particular node.

Data model impact

- A new node field `graphical_console_enabled`, during upgrade/conversion will be populated from `default_graphical_console_interface` config option (`no-graphical-console` by default).
- new node field `graphical_console_interface` will be added.

State Machine Impact

None.

REST API impact

- Add a new optional `console_type` parameter to `GET /v1/nodes/{node_ident}/states/console` and `PUT /v1/nodes/{node_ident}/states/console` APIs. This parameter defines which type of console the Ironic users want to access. The default value will be `serial`. The possible values are as follows:
  - `serial` - For accessing the serial console.
  - `graphical` - For accessing the graphical console.

  This parameter will be included in the query string.

Example:

```
GET /v1/nodes/{node_ident}/states/console?console_type=graphical
```

The response would be same as the console interface. A new 400 HTTP response will be returned if user provides a invalid `console_type`.

The API microversion will need to be bumped.

Client (CLI) impact

- A new option `--type` will be added to OSC command `openstack baremetal node console enable` and `openstack baremetal node console disable`.
- A new option `--type` will be added to OSC command `openstack baremetal node console show`.

RPC API impact

- Add a new `console_type` parameter to `get_console_information`
- Add a new `console_type` parameter to `set_console_mode`

The RPC API microversion will need to be bumped.

Driver API impact

- The new `GraphicalConsoleInterface` will be included in the standardized interfaces group. It is not a mandatory interface.

Nova driver impact

Nova impacts are fully described in the VNC console support for Ironic driver[1] blueprint in Nova.

Essentially, the Ironic virt driver will have to implement `get_vnc_console` and call Ironics get/set-console-mode with the `graphical` type.

As per policy in Nova, changes cannot land until ironic and python-ironicclient changes have landed. The changes on the Nova side are fairly straightforward.

Ramdisk impact

None.

Security impact

- The VNC connection to the nodes are secured by a token generated while creating the console in Nova.
- With standalone Ironic deployment, this will return a URL and a user could directly connect with it. The connection to the baremetal node will not be secure.

Other end user impact

- The number of maximum connections per console, specifically VNC consoles is implementation specific. Some servers are capable of multiple connections and others arent.

Scalability impact

- As mentioned in the last section, the number of connections varies based on the hardware.
- TODO(mkrai): Update the number of connections a conductor can handle to address Rubys comment on PS7.

[1] https://blueprints.launchpad.net/nova/+spec/ironic-vnc-console

Performance Impact

None.

Other deployer impact

- Adds `enabled_graphical_console_interfaces` config option.
- Adds `default_graphical_console_interface` config option.

Developer impact

Driver developers can now offer multiple console interfaces rather than sticking to a single one. This actually maps much better to the reality of servers often offering a Serial-on-LAN access along with a Keyboard-Video-Mouse access.

6.7.3 Implementation

Assignee(s)

Primary assignee:

- mkrai

Other contributors:

- anupn

Work Items

- Introduce `ipmitool.IPMIVNCConsole(BaseInterface)`
- Add `console_type` support to the console REST API.
- Add `console_type` support to the RPC methods.
- Add `console_type` support to the OSC plugin.
- Add graphical console support to VirtualBMC
- Implement basic enable-disable + connect testing within devstack
- Update documents to explain how graphical console can be used

6.7.4 Dependencies

None.

6.7.5 Testing

- Unit tests
- CI testing of `ipmitool.IPMIVNCConsole` with a basic enable-disable connect test.
- Add support for graphical console support in virtual BMC for gate test.

6.7.6 Upgrades and Backwards Compatibility

Proper compatibility with Nova will be ensured. A newer Nova will continue to behave as it currently does when running with an older ironic. A newer ironic will expose features that Nova will simply not use.

Backwards compatibility within ironic is assured through RPC versions. Additional care is taken to ensure out-of-tree drivers are still compatible because the code will specifically handle drivers not switched to the new hardware types. Specific tests covering this part will be added. Finally, compatibility with older API clients is assured through REST API microversions.

6.7.7 Documentation Impact

- Documentation will be updated.

6.7.8 References

CHAPTER

SEVEN

INDICES AND TABLES

- search

BIBLIOGRAPHY

[NMTSTORY] https://storyboard.openstack.org/#!/story/2006506

[NODELEAS] https://opendev.org/openstack/ironic-specs/src/commit/6699db48d78b7a42f90cb5c06ba18a72f94b6667/specs/approved/node-lessee.rst

[APPCREDS] https://docs.openstack.org/keystone/latest/user/application_credentials.html

[SUSHY] https://docs.openstack.org/sushy-tools/latest/

[VIRTBMC] https://docs.openstack.org/project-deploy-guide/tripleo-docs/latest/environments/virtualbmc.html

[PREVSPEC] https://review.opendev.org/c/openstack/ironic-specs/+/764801/3/specs/approved/power-control-passthrough.rst

[RFSHSPEC] https://www.dmtf.org/sites/default/files/standards/documents/DSP0266_1.0.0.pdf

[RFSHSCHM] https://www.dmtf.org/sites/default/files/standards/documents/DSP8010_1.0.0.zip

[RFC7617] https://datatracker.ietf.org/doc/html/rfc7617

[IRONCAPI] https://docs.openstack.org/api-ref/baremetal

[KSTNEAPI] https://docs.openstack.org/api-ref/identity/v3/index.html

[WSGIDISP] https://werkzeug.palletsprojects.com/en/2.0.x/middleware/dispatcher/

[NMTSTORY] https://storyboard.openstack.org/#!/story/2006506

[NODELEAS] https://opendev.org/openstack/ironic-specs/src/commit/6699db48d78b7a42f90cb5c06ba18a72f94b6667/specs/approved/node-lessee.rst

[APPCREDS] https://docs.openstack.org/keystone/latest/user/application_credentials.html

[SUSHY] https://docs.openstack.org/sushy-tools/latest/

[VIRTBMC] https://docs.openstack.org/project-deploy-guide/tripleo-docs/latest/environments/virtualbmc.html

[PREVSPEC] https://review.opendev.org/c/openstack/ironic-specs/+/764801/3/specs/approved/power-control-passthrough.rst

[RFSHSPEC] https://www.dmtf.org/sites/default/files/standards/documents/DSP0266_1.0.0.pdf

[RFSHSCHM] https://www.dmtf.org/sites/default/files/standards/documents/DSP8010_1.0.0.zip

[RFC7617] https://datatracker.ietf.org/doc/html/rfc7617

[IRONCAPI] https://docs.openstack.org/api-ref/baremetal

[KSTNEAPI] https://docs.openstack.org/api-ref/identity/v3/index.html

[WSGIDISP] https://werkzeug.palletsprojects.com/en/2.0.x/middleware/dispatcher/