OpenStack: How do I integrate Ironic with Bright OpenStack 8.2?

NOTE:

The guidance given in the following text is based on an investigation by Bright engineers. The end result is an Ironic integration that proved to be working with Bright Cluster Manager 8.2. However, it should be noted that setting up and running Ironic does not fall under Bright's standard support scope (https://www.brightcomputing.com/support/). This means that while a cluster administrator is expected to be able to set up Ironic on Bright 8.2 with these instructions, Bright support will not be able to provide assistance on any issues related to Ironic under a standard support agreement.

Introduction:

Ironic is an OpenStack project which provisions bare metal (as opposed to virtual) machines. It may be used independently or as part of an OpenStack Cloud, and integrates with the OpenStack Identity (keystone), Compute (nova), Network (neutron), Image (glance), and Object (swift) services.

The Bare Metal service manages hardware through both common (eg. PXE and IPMI) and vendor-specific (eg: Delll's iDrac) remote management protocols. It provides the cloud administrator with a unified interface to a heterogeneous fleet of servers while also providing the Compute service with an interface that allows physical servers to be managed as though they were virtual machines.

This document describes the process of deploying the OpenStack bare metal service (a.k.a Ironic) on a Bright OpenStack cloud. Note that this guide will not explain every detail about the Ironic project. Readers are encouraged to have a look at the upstream documentation, at a minimum the introductory parts of it. This can be found at:

https://docs.openstack.org/ironic/rocky/

This setup is done using Bright OpenStack 8.2 "Rocky" on a standard x86 hardware + a Cisco Nexus 5K switch used for switch integration. The switch can be replaced by another switch as long as there is an ML2 plugin supporting it, this is discussed in detail in "Multi-tenancy in the Bare Metal service" section later in this document.

Only if multi-tenancy "VLAN based projects" is used with Ironic, is switch integration needed.

Otherwise a flat network(s) could be used, which does not require switch integration. The multi-tenancy approach is generally recommended since it is suitable for most production environments.

Note: both can co-exist on the same setup, which we have done in this document.

Bare Metal service components:

The Bare Metal service includes the following components:

ironic-api

A RESTful API that processes application requests by sending them to the ironic-conductor over remote procedure call (RPC). Can be run through WSGI or as a separate process.

ironic-conductor

Adds/edits/deletes nodes; powers on/off nodes with IPMI or other vendor-specific protocol; provisions/deploys/cleans bare metal nodes.

ironic-conductor uses drivers to execute operations on hardware.

ironic-python-agent

A Python service which is run in a temporary ramdisk on the bare metal machine to provide ironic-conductor and ironic-inspector services with remote access, in-band hardware control, and hardware introspection.

Additionally, the Bare Metal service has certain external dependencies, which are very similar to other OpenStack services. The instances of these are by default already configured on a standard Bright OpenStack cluster (they are used by other OpenStack services) -- Ironic will be plugged into these existing instances:

- A database to store hardware information and state. You can set the database back-end type and location. A simple approach is to use the same database backend as the Compute service. Another approach is to use a separate database back-end to further isolate bare metal resources (and associated metadata) from users.
- An oslo.messaging compatible queue, such as RabbitMQ. It may use the same implementation as that of the Compute service, but that is not a requirement. Used to implement RPC between ironic-api and ironic-conductor.

python-ironicclient

A command-line interface (CLI) and python bindings for interacting with the Bare Metal service.

Common Considerations:

- The conductor manages a certain proportion of nodes, distributed to it via a hash ring. This includes constantly polling these nodes for their current power state and hardware sensor data (if enabled and supported by hardware). The IPMI/BMC IPs of the physical servers need to be accessible from the controller/conductor node.
- The conductor needs access to the management controller of each node it manages.
- The conductor co-exists with TFTP (for PXE) and/or HTTP (for iPXE) services that provide the kernel and ramdisk to boot the nodes. The conductor manages them by writing files to their root directories.
- There must be mutual connectivity between the conductor and the nodes being deployed or cleaned.
- By default, the Bare Metal service will pick the smallest hard drive that is larger than 4 GiB for deployment. Another hard drive can be used, but it requires setting root device hints.
- The machines should have enough RAM to fit the deployment/cleaning ramdisk to run. The minimum varies greatly depending on the way the ramdisk was built. For example, tinyipa, the TinyCoreLinux-based ramdisk only needs 400 MiB of RAM, while ramdisks built by diskimage-builder may require 3 GiB or more. The custom deployment image we will build later will require at least 1.5 GB RAM on the bare metal node.
- Partition images are only supported with GNU/Linux operating systems.
- If you plan on using local boot, your partition images must contain GRUB2 bootloader tools to enable Ironic to set up the bootloader during deploy.
- We skipped the HA and scalability configuration in this demo, but you can refer to the upstream documents for more details on this.

Installation:

Setup the database for Ironic

First off, a database needs to be created for Ironic to store its information. The default galera cluster, which is part of Bright OpenStack deployment, can be used for that.

- Get the credentials from the head node, in /root/.galerarc
- Log in to (one of) the OpenStack controller node(s) and run the mysql client:
  mysql -u$MYSQL_USER -p
- Execute the following commands, replacing IRONIC_DBPASSWORD with a suitable password:

```
CREATE DATABASE ironic CHARACTER SET utf8;
GRANT ALL PRIVILEGES ON ironic.* TO 'ironic'@'localhost' \
      IDENTIFIED BY 'IRONIC_DBPASSWORD';
GRANT ALL PRIVILEGES ON ironic.* TO 'ironic'@'%' \
      IDENTIFIED BY 'IRONIC_DBPASSWORD';
```

Install software packages

Some Ironic packages have to be installed into the software image that is used by the conductor node. To do that installation, run the following commands on the head node. Replace ‘default-image’ with a dedicated image you might want to use for Ironic nodes:

```
# yum --disablerepo=cm-rhel7-8.2-updates install python-proliantutils-2.6.0-1.el7.noarch
--installroot=/cm/images/default-image
```

```
# yum install openstack-ironic-api openstack-ironic-conductor python-ironicclient --installroot=/cm/images/default-image
```

It’s convenient to also install the client package onto the head node:

```
# yum install python-ironicclient
```

Configure the Identity service for the Bare Metal service:

-Create the Ironic user and give that user the Admin role
  Note: “IRONIC_PASSWORD” is not the DB’s “IRONIC_DBPASSWORD”, although it could be the same.

$ openstack user create --password IRONIC_PASSWORD ironic
$ openstack role add --project service --user ironic admin

-Create Ironic service

$ openstack service create --name ironic --description "Ironic baremetal provisioning service" baremetal

- Create the endpoints

Note: there was an issue where the ironic IPA (Ironic-Python-Agent) agent inside the deploy image was unable to communicate with the Ironic-conductor using the endpoint's URL due to name resolution issues. If this is the case for you, then you can use IP addresses when you create the endpoints instead of hostnames. Here we are using oshaproxy's IP address:

$ openstack endpoint create --region openstack baremetal admin http://10.141.255.245:6385
$ openstack endpoint create --region openstack baremetal public http://10.141.255.245:6385
$ openstack endpoint create --region openstack baremetal internal http://10.141.255.245:6385

-Create the baremetal roles

$ openstack role create baremetal_admin
$ openstack role create baremetal_observer

-Assign baremetal_admin to ironic user

$ openstack role add --project service --user ironic baremetal_admin

-You can further restrict access to the Bare Metal service by creating a separate “baremetal” Project. This way Bare Metal resources (Nodes, Ports, etc) are only accessible to members of this Project:

$ openstack project create baremetal

-At this point, you may grant read-only access to the Bare Metal service API, without granting any other access, by issuing the following commands:

$ openstack user create --domain default --project-domain default --project baremetal --password PASSWORD USERNAME
$ openstack role add --user-domain default --project-domain default --project baremetal --user

USERNAME baremetal_observer

-Then, install the networking-baremetal plugin and agent with:

chroot /cm/images/default-image

pip install networking-baremetal

-Then, enable baremetal mechanism driver in the Networking service:

To enable the baremetal mechanism drivers in the ML2 plug-in, edit the /etc/neutron/plugins/ml2/ml2_conf.ini configuration file. We will do this via configuration overlay customizations:

[osdev->configurationoverlay[OpenStackControllers]->customizations]% add /etc/neutron/plugins/ml2/ml2_conf.ini

[osdev->configurationoverlay*[OpenStackControllers*]->customizations*[/etc/neutron/plugins/ml2/ml2_conf.ini*]]% entries

[.ml2/ml2_conf.ini*]->entries]% add [ml2]\ mechanism_drivers

[.ml2/ml2_conf.ini*]->entries*[[ml2] mechanism_drivers*]]% set value linuxbridge,openvswitch,l2population,baremetal

[.ml2/ml2_conf.ini*]->entries*[[ml2] mechanism_drivers*]]% show

Parameter Value

--- ---

Enabled yes

Revision

Separator <0 bytes>

Key [ml2] mechanism_drivers

Value linuxbridge,openvswitch,l2population,baremetal

Action Smart Add

[.ml2/ml2_conf.ini*]->entries*[[ml2] mechanism_drivers*]]% commit

Note: The preceding appends “baremetal” to the key=value parameters ([ml2] mechanism_drivers=linuxbridge,openvswitch,l2population) in the ml2_conf.ini file. This can also be done using networking settings mode in the openstack mode in cmsh, as follows:

[cluster->openstack[default]>settings>networking]% set ml2mechanismdrivers baremetal

Configure ironic-neutron-agent:

To configure the baremetal neutron agent, edit the neutron configuration /etc/neutron/plugins/ml2/ironic_neutron_agent.ini file in the software image. Add an [ironic] section. For example:

[ironic]

project_domain_name = Default

project_name = service

user_domain_name = Default

password = <IRONIC_PASSWORD>

username = ironic

auth_url = http://localhost:5000/v3

auth_type = password

region_name = openstack

Enable & Start ironic-neutron-agent service:

Create the new systemd service file “/etc/systemd/system/ironic-neutron-agent.service” in the software image “chroot /cm/images/<*image-name*>” with contents as below:

[Unit]

Description=OpenStack ironic-neutron-agent

After=syslog.target network.target

[Service]

Type=simple

User=neutron

ExecStart=/usr/bin/ironic-neutron-agent --config-dir /etc/neutron --config-file /etc/neutron/plugins/ml2/ironic_neutron_agent.ini --log-file /var/log/neutron/ironic_neutron_agent.log

PrivateTmp=false

KillMode=process

Restart=on-failure

[Install]

WantedBy=multi-user.target

We need to modify the dhcp agent config before starting the service, and remove /etc/neutron/dnsmasq.bright.conf from network nodes:

Note: these steps are not necessary if you are using Bright 8.2-5 or above!

[osdev->configurationoverlay[OpenStackNetworkNodes]->customizations]% add /etc/neutron/dhcp_agent.ini

[osdev->configurationoverlay*[OpenStackNetworkNodes*]->customizations*[/etc/neutron/dhcp_agent.ini*]]% entries

[...*[/etc/neutron/dhcp_agent.ini*]->entries]% add dnsmasq_config_file

[...*[/etc/neutron/dhcp_agent.ini*]->entries*[[DEFAULT] dnsmasq_config_file*]]% set action remove

[...*[/etc/neutron/dhcp_agent.ini*]->entries*[[DEFAULT] dnsmasq_config_file*]]% commit

Bright’s dnsmasq config file /etc/neutron/dnsmasq.bright.conf will cause the ironic agent service to fail. On the controller node freeze the file and remove it as follows:

Modify /cm/local/apps/cmd/etc/cmd.conf and add “/etc/neutron/dnsmasq.bright.conf” to FrozenFile, e.g. FrozenFile = { "/etc/neutron/dnsmasq.bright.conf"}

Then remove the file: # rm /etc/neutron/dnsmasq.bright.conf

Finally, add the service to Bright:

[osdev->device[node002]->services]% add ironic-neutron-agent.service

[osdev->device*[node002*]->services*[ironic-neutron-agent.service*]]% set monitored yes

[osdev->device*[node002*]->services*[ironic-neutron-agent.service*]]% set autostart yes

[osdev->device*[node002*]->services*[ironic-neutron-agent.service*]]% commit

Note: the service will fail to start since the controller node doesn't have the rest of the components configured yet.

Configuring ironic-api & ironic-conductor service

We will need to make the below changes to /etc/ironic/ironic.conf in the software image used by the controller node in the relevant sections:

NOTE: To avoid going through all the below steps, at the end of this section there is the final ironic.conf config file. Replace the one in the software image with it and modify the parameters according to your setup, However beware, some options require installing and configuring certain services, as mentioned in these steps.

Configuring ironic-api service

1. Configure the location of the database via the connection option. In the following, replace IRONIC_DBPASSWORD with the password of your ironic user, and replace DB_IP with the IP address where the DB server is located:

```
[database]
# The SQLAlchemy connection string used to connect to the
# database (string value)
connection=mysql+pymysql://ironic:IRONIC_DBPASSWORD@DB_IP/ironic?charset=utf8
```

2. Configure the ironic-api service to use the RabbitMQ message broker by setting the following option. Replace RPC_* with the appropriate address details and credentials of the RabbitMQ server. If you want to reuse the existing Bright OpenStack RabbitMQ cluster, then get the host/credentials from, e.g., /etc/nova/nova.conf on the controller:

```
[DEFAULT]
# A URL representing the messaging driver to use and its full
# configuration. (string value)
transport_url = rabbit://RPC_USER:RPC_PASSWORD@RPC_HOST:RPC_PORT/
```

3. Configure the ironic-api service to use these credentials with the Identity service. Replace PUBLIC_IDENTITY_IP with the public IP of the Identity server, PRIVATE_IDENTITY_IP with the private IP of the Identity server and replace IRONIC_PASSWORD with the password you chose for the ironic user in the Identity service:

```
[DEFAULT]
# Authentication strategy used by ironic-api: one of
# "keystone" or "noauth". "noauth" should not be used in a
# production environment because all authentication will be
# disabled. (string value)
auth_strategy=keystone
[keystone_authtoken]
# Authentication type to load (string value)
auth_type=password
# Complete public Identity API endpoint (string value)
www_authenticate_uri=http://PUBLIC_IDENTITY_IP:5000
# Complete admin Identity API endpoint. (string value)
auth_url=http://PRIVATE_IDENTITY_IP:5000
# Service username. (string value)
username=ironic
# Service account password. (string value)
password=IRONIC_PASSWORD
# Service tenant name. (string value)
project_name=service
# Domain name containing project (string value)
project_domain_name=Default
# User's domain name (string value)
user_domain_name=Default
```

4. Reboot the controller node or use imageupdate from cmsh to push the changes to the controller node

5. Create the Bare Metal service database tables on the controller node:
   $ ironic-dbsync --config-file /etc/ironic/ironic.conf create_schema

Configuring ironic-conductor service

1. Replace HOST_IP with IP of the conductor host.

```
[DEFAULT]
# IP address of this host. If unset, will determine the IP
# programmatically. If unable to do so, will use "127.0.0.1".
# (string value)
my_ip=HOST_IP
```

Note: If a conductor host has multiple IPs, my_ip should be set to the IP which is on the same network as the bare metal nodes

2. Configure credentials for accessing other OpenStack services.
   In order to communicate with other OpenStack services, the Bare Metal service needs to use service users to authenticate to the OpenStack Identity service when making requests to other services. These users' credentials have to be configured in each configuration file section related to the corresponding service:
   - [neutron] - to access the OpenStack Networking service
   - [glance] - to access the OpenStack Image service
   - [swift] - to access the OpenStack Object Storage service (not installed by default with Bright OpenStack)
   - [inspector] - to access the OpenStack Bare Metal Introspection service (if you choose to install it)
   - [service_catalog] - a special section holding credentials that the Bare Metal service will use to discover its own API URL endpoint as registered in the OpenStack Identity service catalog.

For simplicity, you can use the same service user for all services. For backward compatibility, this should be the same user configured in the [keystone_authtoken] section for the ironic-api service . However, this is not necessary, and you can create and configure separate service users for each service.

```
[neutron]
# Authentication type to load (string value)
auth_type = password
# Authentication URL (string value)
auth_url=https://IDENTITY_IP:5000/
```

```
# Username (string value)
username=ironic
# User's password (string value)
password=IRONIC_PASSWORD
# Project name to scope to (string value)
project_name=service
# Domain ID containing project (string value)
project_domain_id=default-domain-ID
# User's domain id (string value)
user_domain_id=default-domain-ID

# The default region_name for endpoint URL discovery. (string
# value)
region_name = openstack
# List of interfaces, in order of preference, for endpoint
# URL. (list value)
valid_interfaces=public
```

3. Configure enabled drivers and hardware types:

   - Hardware types are enabled in the configuration file of the ironic-conductor service by setting the enabled_hardware_types configuration option, for example:
     [DEFAULT]
     enabled_hardware_types = ipmi,redfish
     Due to the driver's dynamic nature, they also require configuring enabled hardware interfaces.
     Note: All available hardware types and interfaces are listed in setup.cfg file in the source code tree:
     https://git.openstack.org/cgit/openstack/ironic/tree/setup.cfg?h=stable%2Frocky

   - BIOS
     manages configuration of the BIOS settings of a bare metal node. This interface is vendor-specific and can be enabled via the enabled_bios_interfaces option:
     enabled_bios_interfaces = no-bios

   - Boot
     manages booting of both the deployed ramdisk, as well as the user instances on the bare metal node. The boot interface implementations are often vendor specific, and can be enabled via the enabled_boot_interfaces option:
     enabled_boot_interfaces = pxe

     PXE boot interface requires tftp to be set up correctly:
     Set up PXE on the conductor node:

     1. #yum --installroot=/cm/images/default-image install syslinux-tftpboot

2. #cp -a /cm/images/default-image/var/lib/tftpboot/* /cm/images/default-image/tftpboot/

3. There is no need to install and enable the tftp server, it's already included in Bright by default, and socket is activated via systemd.

4. Create a map file:
   echo 're ^(/tftpboot/) /tftpboot/\2' > /cm/images/default-image/tftpboot/map-file
   echo 're ^/tftpboot/ /tftpboot/' >> /cm/images/default-image/tftpboot/map-file
   echo 're ^(^/) /tftpboot/\1' >> /cm/images/default-image/tftpboot/map-file
   echo 're ^([^/]) /tftpboot/\1' >> /cm/images/default-image/tftpboot/map-file

5. Make sure ironic can create files in /tfpboot:
   chroot /cm/images/default-image chown -R ironic /tftpboot

6. Remove /tftpboot from the sync and update exclude lists for the category that the conductor node is in, or the contents of /tftpboot in the image will not be synced.
   [osdev->category[default]]% get excludelistupdate |grep tftpboot

   #no-new-files: - /tftpboot/*

   [osdev->category[default]]% get excludelistsyncinstall |grep tftpboot

   #no-new-files: - /tftpboot/*

   Note: due to an issue with excludelists that should be fixed in Bright 8.2-5, you should scp /tftpboot/ contents from the software image to the controller node (see CM-23091 for details).

- Console
  manages access to the serial console of a bare metal node. We did not configure it in this scenario, but if needed, please refer to: https://docs.openstack.org/ironic/rocky/admin/console.html

- Deploy
  defines how the image gets transferred to the target disk. See Deploy Interfaces for an explanation of the difference between the supported deploy interfaces direct and iscsi.

Note: Direct requires object storage service to be available on the OS deployment which is not installed by default. Hence we enabled iscsi only:
enabled_deploy_interfaces = iscsi
Note: Ensure that the qemu-img and iscsiadm (iscsi-initiator-utils) tools are installed on the controller node (software image) where ironic-conductor will run for iscsi interface to work.

- Inspect
  implements fetching hardware information from nodes. Can be implemented out-of-band (via contacting the node's BMC) or in-band (via booting a ramdisk on a node). The latter implementation is called inspector and uses a separate service called ironic-inspector. *The latter implementation is not configured in this setup*
  enabled_inspect_interfaces = no-inspect,inspector

- Management
  provides additional hardware management actions, like getting or setting boot devices. This interface is usually vendor-specific, and its name often matches the name of the hardware type (with ipmitool being a notable exception). For example:
  enabled_management_interfaces = ipmitool,redfish
  Note: no Bright ipmitool should be installed in the software image. The reason is that Bright's cm-ipmitool installs ipmitool in /cm/local/apps and needs ipmitool module to be loaded first to add it to $PATH.
  You should verify it is working from the controller node, for example:
  ipmitool -I lanplus -H <ip-address> -U <username> -P <password> chassis power status

- Power
  runs power actions on nodes. Similar to the management interface, it is usually vendor-specific, and its name often matches the name of the hardware type (with ipmitool being again an exception). For example:
  enabled_power_interfaces = ipmitool,redfish

- Raid
  manages building and tearing down RAID on nodes. Similar to inspection, it can be implemented either out-of-band or in-band (via agent implementation). For example:
  enabled_raid_interfaces = agent,no-raid
  Note: not used in this setup, but could be used.

- Storage
  manages the interaction with a remote storage subsystem, such as the Block Storage service, and helps facilitate booting from a remote volume. This interface ensures that volume target and connector information is updated during the lifetime of a deployed instance.
  enabled_storage_interfaces = cinder,noop
  Note: not used in this setup, but could be with other dependencies configured!

- Vendor
  is a place for vendor extensions to be exposed in API.
  enabled_vendor_interfaces = ipmitool,no-vendor

- Configuring interface defaults
  These will be used if you did not specify any during baremetal node creation
  default_deploy_interface = iscsi
  default_network_interface=neutron

- Configure the network for ironic-conductor service to perform node cleaning
  [conductor]
  automated_clean = false
  Note: automated cleaning should be set to True in a production environment to avoid data leakage and security-related issues. The downside of this is that it takes some time to do secure disk erasing and nodes can’t be used during the cleaning process. You can speed things up using disks with support for cryptographic ATA Security Erase.

Reboot the controller node & add Ironic services

Add openstack-ironic-conductor & openstack-ironic-api services for the controller node in cmsh to be as follows:

[osdev->device[node002]->services]% show openstack-ironic-conductor

Parameter Value

Autostart yes

Belongs to role no

Monitored yes

Revision

Run if ALWAYS

Service openstack-ironic-conductor

Sickness check interval 60

Sickness check script

Sickness check script timeout 10

Timeout -1

[osdev->device[node002]->services]% show openstack-ironic-api

Parameter Value

---------- ----------

Autostart yes

Belongs to role no

Monitored yes

Revision

Run if ALWAYS

Service openstack-ironic-api

Sickness check interval 60

Sickness check script

Sickness check script timeout 10

Timeout -1

[osdev->device[node002]->services]%

Configure the Compute service to use the Bare Metal service

As of the Newton release, it is possible to have multiple nova-compute services running the ironic virtual driver (in nova) to provide redundancy. Bare metal nodes are mapped to the services via a hash ring. If a service goes down, the available bare metal nodes are remapped to different services.

Once active, a node will stay mapped to the same nova-compute even when it goes down. The node is unable to be managed through the Compute API until the service responsible returns to an active state.

In this setup, we used All the compute nodes “nova-compute” for ironic, which means that you will NOT be able to create regular instances.

OpenStack: How do I integrate Ironic with Bright OpenStack 8.2?

If you want, you can use one or more compute nodes only for Ironic, and leave the rest to be used for creating regular OpenStack instances. To do so, in the following steps, instead of modifying nova.conf in OpenStackHypervisors configuration overlay, you can clone OpenStackHypervisors to a new one, e.g. OpenStackHypervisors-Ironic and add the selected compute nodes to it.

Perform the following changes on the controller node AND compute nodes, and change the values to match your environment:

-Compute/hypervisors:

[osdev->configurationoverlay]% use openstackhypervisors

[osdev->configurationoverlay[OpenStackHypervisors]]% customizations

[osdev->configurationoverlay[OpenStackHypervisors]->customizations]% list

File (key) Type Label Enabled

---------- ---------- ---------- -------

/etc/nova/nova.conf INI File yes

[osdev->configurationoverlay[OpenStackHypervisors]->customizations]% use /etc/nova/nova.conf

[osdev->configurationoverlay[OpenStackHypervisors]->customizations[/etc/nova/nova.conf]]% entries

[osdev->configurationoverlay[OpenStackHypervisors]->customizations[/etc/nova/nova.conf]->entries]% ls

Key (key) Value Action Enabled

---------- ---------- ---------- -------

[DEFAULT] compute_driver ironic.IronicDriver Smart Add yes

[DEFAULT] reserved_host_memory_mb 0 Smart Add yes

[filter_scheduler] track_instance_changes false Smart Add yes

[ironic] auth_type password Smart Add yes

[ironic] auth_url http://10.141.255.245:5000/v3 Smart Add yes

[ironic] password system Smart Add yes

[ironic] project_domain_name default Smart Add yes

[ironic] project_name service Smart Add yes

[ironic] user_domain_name default Smart Add yes

[ironic] username ironic Smart Add yes

[scheduler] discover_hosts_in_cells_interval 120 Smart Add yes

-Controller nodes:

[osdev]% configurationoverlay

[osdev->configurationoverlay]% use openstackcontrollers

[osdev->configurationoverlay[OpenStackControllers]]% customizations

[osdev->configurationoverlay[OpenStackControllers]->customizations]% ls

File (key) Type Label Enabled

------------------------------------ ---- ------ -------

/etc/neutron/plugins/ml2/ml2_conf.ini INI File yes

/etc/nova/nova.conf INI File yes

[osdev->configurationoverlay[OpenStackControllers]->customizations]% use /etc/nova/nova.conf

[osdev->configurationoverlay[OpenStackControllers]->customizations[/etc/nova/nova.conf]]% entries

[osdev->configurationoverlay[OpenStackControllers]->customizations[/etc/nova/nova.conf]->entries]% ls

Key (key) Value Action Enabled

---------- ---------- ---------- -------

[DEFAULT] compute_driver ironic.IronicDriver Smart Add yes

[DEFAULT] reserved_host_memory_mb 0 Smart Add yes

[filter_scheduler] track_instance_changes false Smart Add yes

[ironic] auth_type password Smart Add yes

[ironic] auth_url http://10.141.255.245:5000/v3 Smart Add yes

[ironic] password system Smart Add yes

[ironic] project_domain_name default Smart Add yes

[ironic] project_name service Smart Add yes

[ironic] user_domain_name default Smart Add yes

[ironic] username ironic Smart Add yes

[scheduler] discover_hosts_in_cells_interval 120 Smart Add yes

-Cmdaemon should restart Nova services on the controller and compute nodes after applying the configuration changes.

Configure the Networking service for bare metal provisioning:

You need to configure Networking so that the bare metal server can communicate with the Networking service for DHCP, PXE boot and other requirements.

You will need to provide Bare Metal service with the MAC address(es) of each node that it is provisioning. The Bare Metal service in turn will pass this information to the Networking service for DHCP and PXE boot configuration.

Note: When the baremetal ML2 components are not used, ports in the Networking service will have status: DOWN, and binding_vif_type:binding_failed. This was always the status for Bare Metal service flat network interface ports prior to the introduction of the baremetal ML2 integration. For a non-routed network, bare metal servers can still be deployed and are functional, despite this port binding state in the Networking service.

-A big part of the integration here was done in previous steps, with the exception of below, while the rest is taken care of by cmdaemon:

in ml2_conf.ini the configuration should look like the following in the configuration overlay:

[osdev->configurationoverlay[OpenStackControllers]->customizations[/etc/neutron/plugins/ml2/ml2_conf.ini]->entries]% ls

Key (key) Value Action Enabled

---------- ---------- --------- -------

[ml2] mechanism_drivers linuxbridge,openvswitch,l2population,baremetal,genericswitch Smart Add yes

Adding the mechanism drivers for baremetal and genericswitch (as done in the preceding) can instead be done using networking settings mode in the openstack mode in cmsh as follows, which appends the drivers by default to the ml2_conf.ini file:

[cluster->openstack[default]>settings>networking]% set ml2mechanismdrivers baremetal,genericswitch

-We also assume the default flat network “bright-internal-flat-internalnet” created by Bright is available. This network gets automatically created as part of deploying Bright OpenStack.

-DHCP needs to be enabled on the flat network:

[root@osdev ~]# openstack subnet set --dhcp bright-internal-flat-internalnet-sn

-DHCP needs to be locked on the internalnet in cmsh. Otherwise if you try to provision a baremetal node on the flat internalnet network, it might boot from the head node:

[osdev]% network
[osdev->network]% use internalnet
[osdev->network[internalnet]]% set lockdowndhcpd yes
[osdev->network[internalnet*]]% commit

-Note: you will need also to make sure that the physical node you are going to provision is not added to Bright or at least doesn’t have a MAC address defined in Bright.

OpenStack: How do I integrate Ironic with Bright OpenStack 8.2?

Configure the Bare Metal service for cleaning:

When hardware is recycled from one workload to another, Ironic performs automated cleaning on the node to ensure it is ready for another workload. This ensures that the tenant will get a consistent bare metal node deployed every time. Refer to the upstream documents (at https://docs.openstack.org/ironic/rocky/admin/cleaning.html) for details.
Configure the cleaning network UUID via the cleaning_network option in the Bare Metal service configuration file (/etc/ironic/ironic.conf) in the software image. In the following, replace NETWORK_UUID with the default flat network “bright-internal-flat-internalnet” UUID:

```
[neutron]
cleaning_network = NETWORK_UUID
```

Configure tenant networks:

1. Define a provider network in the Networking service, which we shall refer to as the “provisioning” network. Using the neutron network interface requires that provisioning_network and cleaning_network configuration options are set to valid identifiers (UUID or name) of networks in the Networking service. If these options are not set correctly, then cleaning or provisioning will fail to start. There are two ways to set these values:

- Under the neutron section of ironic configuration file:

```
[neutron]
cleaning_network = $CLEAN_UUID_OR_NAME
provisioning_network = $PROVISION_UUID_OR_NAME
```

- Under provisioning_network and cleaning_network keys of the node’s driver_info field as driver_info['provisioning_network'] and driver_info['cleaning_network'] respectively.

Here we are using the default flat network “bright-internal-flat-internalnet”.

Note: Spawning a bare metal instance onto the provisioning network is impossible -- the deployment will fail. The node should be deployed onto a different network from the provisioning network. When you boot a bare metal instance from the Compute service, you should choose a different network in the Networking service for your instance.

2. The “provisioning” and “cleaning” networks may be the same network or distinct networks. To ensure that communication between the Bare Metal service and the deploy

ramdisk/image works, it is important to ensure that security groups are disabled for these networks, or that the default security groups allow:

- DHCP
- TFTP
- egress port used for the Bare Metal service (6385 by default)
- ingress port used for ironic-python-agent (9999 by default)
- if using iSCSI deploy --- which we do in this setup -- the ingress port used for iSCSI (3260 by default)

In our case the internal net is trusted, so we allowed all ingress TCP, UDP, and ICMP in the security group default in project bright. The rules should look like:

[root@osdev ~]# openstack security group rule list a8ce6608-3abc-4288-ab2e-ea4bb7ef4e17 --long

+--------------------------------------+-------------+-----------+------------+-----------+-----------+--------------------------------------+

| ID | IP Protocol | IP Range | Port Range | Direction | Ethertype | Remote Security Group |

+--------------------------------------+-------------+-----------+------------+-----------+-----------+--------------------------------------+

| 27077681-8c96-463d-9065-ab2162803010 | None | None | | egress | IPv4 | None |

| 4bace51e-fd76-404d-bc1b-4896638100cf | icmp | 0.0.0.0/0 | | ingress | IPv4 | None |

| 5ee755ba-12bd-4906-b0c6-a3103192a7ac | tcp | 0.0.0.0/0 | 1:65535 | ingress | IPv4 | None |

| 7647edb4-3af7-4480-9da1-9eccda1bfcba | None | None | | ingress | IPv6 | a8ce6608-3abc-4288-ab2e-ea4bb7ef4e17 |

| ab69c7e7-a1b0-4417-91d1-77c6696d40b1 | udp | 0.0.0.0/0 | 1:65535 | ingress | IPv4 | None |

| c98f4adb-34ea-4c02-9855-14c0b67c304f | None | None | | egress | IPv6 | None |

| e975db1a-08eb-4645-8836-9e909bf5e952 | None | None | | ingress | IPv4 | a8ce6608-3abc-4288-ab2e-ea4bb7ef4e17 |

+--------------------------------------+-------------+-----------+------------+-----------+-----------+--------------------------------------+

----------+

3. The previous configuration allows for provisioning on Flat networks, assuming the bare-metal node will be attached to the provisioning network during its entire life cycle.

4. To use the concept of Multi-tenancy; you will need to Install and configure a compatible ML2 mechanism driver which supports bare metal provisioning for your switch. In our case the “Networking Generic Switch” is used

Multi-tenancy in the Bare Metal service

Network interface is one of the driver interfaces that manages network switching for nodes. Refer to the upstream document at https://docs.openstack.org/ironic/rocky/install/enabling-drivers.html for details.
There are 3 network interfaces available in the Bare Metal service:

- The noop interface is used for standalone deployments, and does not perform any network switching, This is not used in this setup.

- The flat interface places all nodes into a single provider network that is pre-configured on the Networking service and physical equipment. Nodes remain physically connected to this network during their entire life cycle.

- The neutron interface provides tenant-defined networking through the Networking service, separating tenant networks from each other and from the provisioning and cleaning provider networks. Nodes will move between these networks during their life cycle. This interface requires Networking service support for the switches attached to the bare metal servers so they can be programmed.

Local link connection

The Bare Metal service allows local_link_connection information to be associated with Bare Metal ports. This information is provided to the Networking service’s ML2 driver when a Virtual Interface (VIF) is attached. The ML2 driver uses the information to plug the specified port to the tenant network.

Field	Description
switch_id	Required. Identifies a switch and can be a MAC address or an OpenFlow-based datapath_id.

port_id	Required. Port ID on the switch, for example, Gig0/1.
switch_info	Optional. Used to distinguish different switch models or other vendor-specific identifier. Some ML2 plugins may require this field.

Note: This isn't applicable to Infiniband ports because the network topology is discoverable by the Infiniband Subnet Manager. If specified, local_link_connection information will be ignored.

To install the "Networking Generic Switch":

```
[root@osdev ~]# yum install python2-networking-generic-switch.noarch
--installroot=/cm/images/default-image/
```

To enable the "genericswitch" mechanism driver in Neutron, add "genericswitch" to the mechanism_drivers in neutron in configuration overlay as below:

```
[osdev->configurationoverlay[OpenStackControllers]->customizations[/etc/neutron/plugins/ml2/
ml2_conf.ini]->entries]% ls

Key (key)             Value                                                    Action    Enabled

---------------------- ------------------------------------------------------------------ --------- -------

[...]

[ml2] mechanism_drivers  linuxbridge,openvswitch,l2population,baremetal,genericswitch
    Smart Add yes
```

Adding a mechanism driver can instead be done using networking settings mode in the openstack mode in cmsh, as follows, so that "baremetal" and "genericswitch" are appended by default to the mechanism_drivers entry in the preceding ini file:

```
[cluster->openstack[default]>settings>networking]% set ml2mechanismdrivers
baremetal,genericswitch
```

After that, you will need to add the Switch so it can be managed by neutron, these configuration depends on the switch make and model, refer to the below URL for details:

https://docs.openstack.org/networking-generic-switch/rocky/configuration.html

These configurations are committed to the ml2_config.ini file in the configuration overlay. The final version of the file is shown next. The values shown here should be changed to match your switch:

```
[osdev->configurationoverlay]% use openstackcontrollers

[osdev->configurationoverlay[OpenStackControllers]]% customizations

[osdev->configurationoverlay[OpenStackControllers]->customizations]% ls

File (key)                          Type Label  Enabled

----------------------------------- ---- ------ -------

/etc/neutron/plugins/ml2/ml2_conf.ini  INI  File   yes

/etc/nova/nova.conf                 INI  File   yes

[osdev->configurationoverlay[OpenStackControllers]->customizations]% use
/etc/neutron/plugins/ml2/ml2_conf.ini

[osdev->configurationoverlay[OpenStackControllers]->customizations[/etc/neutron/plugins/ml2/
ml2_conf.ini]]% entries

[...ml2_conf.ini]->entries]% add [genericswitch:nexus5k]\ device_type

[...ml2_conf.ini*]->entries*[[genericswitch:nexus5k] device_type*]]% set value netmiko_cisco_ios

[...ml2_conf.ini*]->entries*[[genericswitch:nexus5k] device_type*]]% ..

[...ml2_conf.ini*]->entries*]% add [genericswitch:nexus5k]\ ngs_mac_address

[...ml2_conf.ini*]->entries*[[genericswitch:nexus5k] ngs_mac_address*]]% set value
54:7f:ee:8c:4b:41

[...ml2_conf.ini*]->entries*[[genericswitch:nexus5k] ngs_mac_address*]]% ..

[...ml2_conf.ini*]->entries*]% add [genericswitch:nexus5k]\ username

[...ml2_conf.ini*]->entries*[[genericswitch:nexus5k] username*]]% set value admin

[...ml2_conf.ini*]->entries*[[genericswitch:nexus5k] username*]]% ..

[...ml2_conf.ini*]->entries*]% add [genericswitch:nexus5k]\ password

[...ml2_conf.ini*]->entries*[[genericswitch:nexus5k] password*]]% set value XXXXXXXXX
```

[...ml2_conf.ini*]->entries*[[genericswitch:nexus5k] password*]]% ..

[...ml2_conf.ini*]->entries*]% add [genericswitch:nexus5k]\ ip

[...ml2_conf.ini*]->entries*[[genericswitch:nexus5k] ip*]]% set value 10.2.160.1

[...ml2_conf.ini*]->entries*[[genericswitch:nexus5k] ip*]]% ..

[...ml2_conf.ini]->entries]% use [ml2]\ mechanism_drivers

[...ml2_conf.ini]->entries[[ml2] mechanism_drivers]]% set value linuxbridge,openvswitch,l2population,baremetal,genericswitch

[...ml2_conf.ini*]->entries*[[ml2] mechanism_drivers*]]% commit

[...ml2_conf.ini]->entries[[ml2] mechanism_drivers]]% ..

[...ml2_conf.ini]->entries]% ls

Key (key) Value Action Enabled

---------- ---------- --------- -------

[genericswitch:nexus5k] device_type netmiko_cisco_ios Smart Add yes

[genericswitch:nexus5k] ip 10.2.160.1 Smart Add yes

[genericswitch:nexus5k] ngs_mac_address 54:7f:ee:8c:4b:41 Smart Add yes

[genericswitch:nexus5k] password ny1x88eT Smart Add yes

[genericswitch:nexus5k] username admin Smart Add yes

[ml2] mechanism_drivers linuxbridge,openvswitch,l2population,baremetal,genericswitch Smart Add yes

[securitygroup] firewall_driver neutron.agent.linux.iptables_firewall.OVSHybridIptablesFirewall+ Smart Add yes

[osdev->configurationoverlay[OpenStackControllers]->customizations[/etc/neutron/plugins/ml2/ml2_conf.ini]->entries]%

To test out the connectivity to the physical switch, use the test script below, adjust to match your

HW switch:

```
[root@node002 ~]# cat ./testCiscoswitch.py

#!/usr/bin/python

from netmiko import ConnectHandler

nexus5k = {

    'device_type': 'cisco_ios',

    'ip':   '10.2.160.1',

    'username': 'admin',

    'password': 'XXXXXXX',

    'verbose': True       # optional, defaults to False

}

net_connect = ConnectHandler(**nexus5k)

output = net_connect.send_command('show vlan')

print(output)
```

Create and add images to the Image service

Bare Metal provisioning requires two sets of images: the deploy images and the user images. The deploy images are used by the Bare Metal service to prepare the bare metal server for actual OS deployment. The user images, on the other hand, are installed on the bare metal server to be used by the end user

Creating user images:

These images are based on the CentOS public cloud image. Here we create a partition image as opposed to whole disk image, because this gives us more flexibility in customizing the bare metal instance that will be created later on.

Install diskimage-builder:

```
[root@osdev ~]# yum install diskimage-builder
```

Create the image, enable DHCP on all interfaces and install GRUB. This will generate 3 images (kernel, initrd and the image), which are used to create the final glance image as shown next:

[root@osdev ~]# disk-image-create centos7 baremetal dhcp-all-interfaces grub2 -o my-image

[root@osdev ~]# glance image-create --name my-kernel --visibility public --disk-format aki --container-format aki < my-image.vmlinuz

UUID of the uploaded image: 6ffb9b36-ea98-44c0-93c2-9ff5a7151a91

[root@osdev ~]# glance image-create --name my-image.initrd --visibility public --disk-format ari --container-format ari < my-image.initrd

UUID of the uploaded image: 4fbfe9be-f6e8-47be-ad59-99cc7b5609f8

[root@osdev ~]# glance image-create --name my-image --visibility public --disk-format qcow2 --container-format bare --property kernel_id=6ffb9b36-ea98-44c0-93c2-9ff5a7151a91 --property ramdisk_id=4fbfe9be-f6e8-47be-ad59-99cc7b5609f8 < my-image.qcow2

Creating deploy images:

Ironic depends on having an image with the ironic-python-agent (IPA) service running on it for controlling and deploying bare metal nodes.

-You can download a pre-built version of the deploy ramdisk built with the CoreOS tools at:

- CoreOS deploy kernel
- CoreOS deploy ramdisk

[root@osdev ~]# wget -4 https://tarballs.openstack.org/ironic-python-agent/coreos/files/coreos_production_pxe-stable-rocky.vmlinuz

[root@osdev ~]# wget -4 https://tarballs.openstack.org/ironic-python-agent/coreos/files/coreos_production_pxe_image-oem-stable-rocky.cpio.gz

[root@osdev ~]# glance image-create --name deploy-vmlinuz --visibility public --disk-format aki --container-format aki < coreos_production_pxe.vmlinuz

[root@osdev ~]# glance image-create --name deploy-initrd --visibility public --disk-format ari --container-format ari < coreos_production_pxe_image-oem.cpio.gz

-The other option would be building your own deploy image, which is recommended. For example, we had a name resolution issue during cleaning and provisioning that we could not

solve without access to the deploy image to troubleshoot, which by default is not possible.

- To do this, we will use the devuser element while building our new deploy image. First export the following environment variables. Change their values appropriately:

```
[root@osdev ~]# export DIB_DEV_USER_PASSWORD=system

[root@osdev ~]# export DIB_DEV_USER_PWDLESS_SUDO=yes

[root@osdev ~]# export DIB_DEV_USER_USERNAME=bright

[root@osdev ~]# export DIB_DEV_USER_AUTHORIZED_KEYS=$HOME/.ssh/id_{rsa,dsa}.pub
```

- Then build the image. This will create an image with the user and password as defined in the above environment variables, so that you can SSH to it for troubleshooting later on:

```
[root@osdev ~]# disk-image-create ironic-agent devuser centos7 -o ironic-deploy 2>&1 |tee
ironic-deploy.log
```

The above command creates the deploy ramdisk and kernel named ironic-deploy.vmlinuz and ironic-deploy.initramfs in your current directory.

- The final step is to upload the images:

```
[root@osdev ~]# glance image-create --name deploy-initrd-password4 --visibility public
--disk-format ari --container-format ari < ironic-deploy-password4.initramfs

[root@osdev ~]# glance image-create --name deploy-vmlinuz-password4 --visibility public
--disk-format aki --container-format aki < ironic-deploy-password4.vmlinuz
```

Create flavors for use with the Bare Metal service

You will need to create a special bare metal flavor in the Compute service. The flavor is mapped to the bare metal node through the node’s resource_class field (available starting with Bare Metal API version 1.21). A flavor can request exactly one instance of a bare metal resource class.

Note that when creating the flavor, it is useful to add the RAM_MB and CPU properties as a convenience to users, although they are not used for scheduling. The DISK_GB property is also not used for scheduling, but is still used to determine the root partition size.

1. Change these to match your hardware:

```
$ RAM_MB=32000
$ CPU=8
$ DISK_GB=200
```

2. Create the bare metal flavor by executing the following command:
   ```
   $ openstack flavor create --ram $RAM_MB --vcpus $CPU --disk $DISK_GB \
    my-baremetal-flavor
   ```

After creation, associate each flavor with one custom resource class or traits, in our case we are using traits:

Starting with the Queens release, the Compute service supports scheduling based on qualitative attributes using traits. Starting with Bare Metal REST API version 1.37, it is possible to assign a list of traits to each bare metal node. Traits assigned to a bare metal node will be assigned to the corresponding resource provider in the Compute service placement API.

When creating a flavor in the Compute service, required traits may be specified via flavor properties. The Compute service will then schedule instances only to bare metal nodes with all of the required traits.

Traits can be either standard or custom. Standard traits are listed in the os_traits library. Custom traits must meet the following requirements:

- prefixed with CUSTOM_
- contain only upper case characters A to Z, digits 0 to 9, or underscores
- no longer than 255 characters in length
- A bare metal node can have a maximum of 50 traits.

[root@osdev ~]# openstack flavor set --property trait:CUSTOM_TRAIT1=required my-baremetal-flavor-with-resources

Enrollment

After all the services have been properly configured, you should enroll your hardware (which will be used for hosting bare metal instances) with the Bare Metal service, and confirm that the Compute service sees the available hardware. The nodes will be visible to the Compute service once they are in the available provision state.

Note: After enrolling nodes with the Bare Metal service, the Compute service will not be immediately notified of the new resources. The Compute service’s resource tracker syncs periodically, and so any changes made directly to the Bare Metal service’s resources will become visible in the Compute service only after the next run of that periodic task.

We are using the IPMI driver in our setup, the driver you use will depend on your hardware.

Use the below commands to list available drivers and show the required and optional properties for the IPMI driver:

```
[root@osdev ~]# openstack baremetal driver list

+---------------------+--------------------+

| Supported driver(s) | Active host(s)     |

+---------------------+--------------------+

| ipmi                | node002.cm.cluster |

| redfish             | node002.cm.cluster |

+---------------------+--------------------+

[root@osdev ~]# openstack baremetal driver property list ipmi --fit-width

+-----------------------------+-----------------------------------------------------------------------------------------------------------------------------------------------------------+

| Property                    | Description                                                                                                                                               |

+-----------------------------+-----------------------------------------------------------------------------------------------------------------------------------------------------------+

| deploy_forces_oob_reboot    | Whether Ironic should force a reboot of the Node via the out-of-band channel after deployment is complete. Provides compatibility with older deploy ramdisks. Defaults to False. Optional. |

| deploy_kernel               | UUID (from Glance) of the deployment kernel. Required.                                                                                                    |

| deploy_ramdisk              | UUID (from Glance) of the ramdisk that is mounted at boot time. Required.                                                                                 |
```

| force_persistent_boot_device | True to enable persistent behavior when the boot device is set during deploy and cleaning operations. Defaults to False. Optional. |

| ipmi_address | IP address or hostname of the node. Required. |

[.....]

| ipmi_username | username; default is NULL user. Optional. |

+------------------------------+--+

Note: different API versions support different features, which means that if a feature is not working it could be that you are not using/exporting the correct API version.

Enrollment process:

1. Creating the node

Create a node in the Bare Metal service with the node create command. At a minimum, you must specify the driver name, ipmi, in this case.
[root@osdev ~]# openstack baremetal node create --driver ipmi
[root@osdev ~]# NODE_UUID=bfedf771-6824-47b7-aa52-a3836ad20eb7

2. Set the node's name and add IPMI information:
   [root@osdev ~]# openstack baremetal node set $NODE_UUID --name node7
   [root@osdev ~]# openstack baremetal node set $NODE_UUID --driver-info ipmi_username=ADMIN --driver-info ipmi_password=ADMIN --driver-info ipmi_address=10.2.186.1

3. Add the UUID of the deploy_kernel and deploy_ramdisk to the driver info. These are the UUIDs from either the images we created earlier using disk-image-builder, or from the downloaded CoreOS images.

4. [root@osdev ~]# openstack baremetal node set $NODE_UUID --driver-info deploy_kernel=75040fa5-41bb-45e3-a501-63f6f0d4466e --driver-info deploy_ramdisk=93ef1217-1886-41cd-a5eb-af329b73f2a4

5. Here we will add the actual physical specs of the node. These could be filed

automatically by ironic-inspector if you choose to install and configure it
[root@osdev ~]# openstack baremetal node set $NODE_UUID --property cpus=8 --property cpu_arch=x86_64 --property memory_mb=32072 --property local_gb=200

6. Add the trait we added to the flavor to match this node
   [root@osdev ~]# openstack --os-baremetal-api-version 1.37 baremetal node add trait $NODE_UUID CUSTOM_TRAIT1

7. Create a baremetal port and add the physical port information to it
   [root@osdev ~]# openstack baremetal port create AC:1F:6B:72:4F:24 --node bfedf771-6824-47b7-aa52-a3836ad20eb7
   [root@osdev ~]# openstack baremetal port set 51c9216f-d449-4225-b2c3-84e98a653af5 --local-link-connection switch_id=54:7f:ee:8c:4b:41 --local-link-connection switch_info=nexus5k --local-link-connection port_id=Eth1/2

8. We now need to validate all the information:
   [root@osdev ~]# openstack baremetal node validate $NODE_UUID
   +------------+--------+--+
   | Interface | Result | Reason |
   +------------+--------+--+
bios	False	Driver ipmi does not support bios (disabled or not implemented).
boot	True	
console	False	Driver ipmi does not support console (disabled or not implemented).
deploy	True	
inspect	True	
management	True	
network	True	
power	True	
raid	False	Driver ipmi does not support raid (disabled or not implemented).
rescue	False	Driver ipmi does not support rescue (disabled or not implemented).
storage	True	
   +------------+--------+--+
   Note: When using the Compute Service with the Bare Metal service, it is safe to ignore the deploy interface’s validation error due to lack of image information. You may continue the enrollment process. This information will be set by the Compute Service just before deploying, when an instance is requested

9. In order for nodes to be available for deploying workloads on them, nodes must be in the available provision state. To do this, nodes created with API version 1.11 and above

must be moved from the enroll state to the manageable state and then to the available state:
[root@osdev ~]# openstack baremetal node manage $NODE_UUID
[root@osdev ~]# openstack baremetal node show $NODE_UUID

10. Now you can move the node to be in the available state:
    [root@osdev ~]# openstack baremetal node provide $NODE_UUID
    [root@osdev ~]# openstack baremetal node show $NODE_UUID

11. Check the node's information from the above command and make sure that it is in the desired state, some values/options are set based on the default ones we set before in ironic.conf, could be overridden in each node if desired, see the defaults again for ipmi driver, run:
    [root@osdev ~]# openstack baremetal --os-baremetal-api-version 1.31 driver show ipmi

```
+------------------------------+----------------------+
| Field                        | Value                |
+------------------------------+----------------------+
| default_boot_interface       | pxe                  |
| default_console_interface    | no-console           |
| default_deploy_interface     | iscsi                |
| default_inspect_interface    | inspector            |
| default_management_interface | ipmitool             |
| default_network_interface    | neutron              |
| default_power_interface      | ipmitool             |
| default_raid_interface       | no-raid              |
| default_vendor_interface     | ipmitool             |
| enabled_boot_interfaces      | pxe                  |
| enabled_console_interfaces   | no-console           |
| enabled_deploy_interfaces    | iscsi                |
| enabled_inspect_interfaces   | inspector, no-inspect |
| enabled_management_interfaces | ipmitool            |
| enabled_network_interfaces   | flat, noop, neutron  |
| enabled_power_interfaces     | ipmitool             |
| enabled_raid_interfaces      | no-raid, agent       |
| enabled_vendor_interfaces    | ipmitool, no-vendor  |
| hosts                        | node002.cm.cluster   |
| name                         | ipmi                 |
| type                         | dynamic              |
+------------------------------+----------------------+
```

12. NOTE: There is a bug in the generic switch plugin which assumes that if you are using a flat network - which we used for provisioning and cleaning - that it tjem uses the default VLAN ID "1" on the physical Cisco switch. This is not true on most setups because you can set a port in access mode with the VLAN ID assigned to this setup, which is not 1 in most cases.

For example, in our setup the VLAN ID is “30”. A quick hack is to change the default VLAN ID from 1 to 30 manually in the software image, and then sync it to the controller node:

```
[root@node002 ~]# vim
/usr/lib/python2.7/site-packages/networking_generic_switch/generic_switch_mech.py

461         # If segmentation ID is None, set vlan 1
462         ###Modified to match VLAN 30 in the lab##segmentation_id =
segments[0].get('segmentation_id') or '1'
463         segmentation_id = segments[0].get('segmentation_id') or '30'
464         provisioning_blocks.add_provisioning_component(
465             context._plugin_context, port['id'], resources.PORT,
466             GENERIC_SWITCH_ENTITY)
467         LOG.debug("Putting port {port} on {switch_info} to vlan: "
468                   "{segmentation_id}".format(
469                       port=port_id,
470                       switch_info=switch_info,
471                       segmentation_id=segmentation_id))
```

If you do not do this the node will later get stuck in the “waiting for agent_callback” state, checking the physical port, and you will find the plugin changed the port to VLAN ID 1 which is incorrect. Unless, of course, you actually are using VLAN ID 1.

13. Now the node is available and ready

Note: the above steps assume that you are going to use multi-tenant networking where the node will be moving between different VLANs during its lifecycle, which requires physical switch integration

Instead of that, you can use flat networks. But in this case the provisioning/cleaning network would be attached to the node all the time. To use flat networks, you will need to change the “network_interface” in the node to be “flat” instead of “neutron”. Also the baremetal port doesn’t need all the link information:

```
[root@osdev ~]# openstack baremetal node set --network-interface flat $NODE_UUID
```

Deploy an instance on the baremetal node

Putting it all together, you will need to create an openstack instance, with the flavor we created before, to match the trait we set on the node. This will use the user-image previously created, and use a network matching the network_interface we choose.

In the case of a flat network, we can use the “bright-internal-flat-internalnet” which is used also for provisioning and cleaning.

If you are using neutron, create a VLAN-based network in OpenStack with the correct segment, matching what is configured on the network switch. Make sure the network node has access to this network as usual so the instance can get the network and metadata information from neutron services running on this node.

Here we are creating a provider network with segment “VLAN” ID of 800, with an appropriate subnet and DHCP enabled:

```
[root@osdev ~]# openstack network create --provider-network-type vlan
--provider-physical-network phyvlanhostnet --provider-segment 800 --share net800

[root@osdev ~]# openstack subnet create --subnet-pool 192.168.0.0/24 --dhcp --network
net800 subnet800
```

Now, create the instance:

```
[root@osdev ~]# openstack server create --flavor 3fc79309-a5c7-4762-9190-e0f568c1510e
--image my-image --key-name keypair --nic net-id=cb4bc78c-1e88-4004-84ff-4929b13a55ab
bare6

[root@osdev ~]# openstack server show bare6 --fit-width

+-------------------------------------+------------------------------------------------+

| Field                               | Valu                                           |

+-------------------------------------+------------------------------------------------+

| OS-DCF:diskConfig                   | MANUAL                                         |

| OS-EXT-AZ:availability_zone         | default                                        |

| OS-EXT-SRV-ATTR:host                | node004                                        |

| OS-EXT-SRV-ATTR:hypervisor_hostname | bfedf771-6824-47b7-aa52-a3836ad20eb7
        |
```

| OS-EXT-SRV-ATTR:instance_name | instance-00000013 |

| OS-EXT-STS:power_state | Running |

| OS-EXT-STS:task_state | None |

| OS-EXT-STS:vm_state | active |

| OS-SRV-USG:launched_at | 2019-01-14T14:03:13.000000 |

| OS-SRV-USG:terminated_at | None |

| accessIPv4 | |

| accessIPv6 | |

| addresses | net800=192.168.100.8 |

| config_drive | |

| created | 2019-01-14T13:58:06Z |

| flavor | my-baremetal-flavor-with-resources |

| | (3fc79309-a5c7-4762-9190-e0f568c1510e) |

| hostId | d1c129b368bf7158f5eb493110939fb7c35c9c28cddfe54b |

| | 680e9e66 |

| id | 5183bec6-112c-482c-bfe7-3a9f2fb97af5 |

| image | my-image (3601e669-f11b-46c4-8714-4895be15f59e) |

| key_name | keypair |

| name | bare6 |

| progress | 0 |

| project_id | cd73c4dc838d4c36bd00dd7bb8fc958a |

| properties | |

| security_groups | name='default' |

| status | ACTIVE |

```
| updated                | 2019-01-14T14:03:13Z                 |
| user_id                | 719dc9025aa94e42a740cff5309f6744     |
| volumes_attached       |                                      |
+------------------------+--------------------------------------+
```

[root@osdev ~]# openstack baremetal node show bfedf771-6824-47b7-aa52-a3836ad20eb7 --fit-width

```
+------------------------+------------------------------------------------------------------------+
| Field                  | Value                                                                  |
+------------------------+------------------------------------------------------------------------+
| bios_interface         | no-bios                                                                |
| boot_interface         | pxe                                                                    |
| chassis_uuid           | None                                                                   |
| clean_step             | {}                                                                     |
| conductor_group        |                                                                        |
| console_enabled        | False                                                                  |
| console_interface      | no-console                                                             |
| created_at             | 2019-01-11T17:39:04+00:00                                              |
| deploy_interface       | iscsi                                                                  |
| deploy_step            | {}                                                                     |
| driver                 | ipmi                                                                   |
| driver_info            | {u'deploy_kernel': u'75040fa5-41bb-45e3-a501-63f6f0d4466e',            |
|                        | u'ipmi_address': u'10.2.186.1', u'ipmi_username': u'ADMIN',            |
|                        | u'ipmi_password': u'******', u'deploy_ramdisk':                        |
|                        | u'93ef1217-1886-41cd-a5eb-af329b73f2a4'}                               |
```

```
| driver_internal_info   | {u'deploy_boot_mode': u'bios', u'is_whole_disk_image': False, |
|                        | u'root_uuid_or_disk_id':                                      |
|                        | u'70cc928b-1750-4675-8180-f2353dc17c2f', u'agent_url':        |
|                        | u'http://10.141.152.1:9999', u'deploy_steps': None,           |
|                        | u'agent_version': u'3.5.1.dev11'}                             |
| extra                  | {}                                                            |
| fault                  | None                                                          |
| inspect_interface      | inspector                                                     |
| inspection_finished_at | None                                                          |
| inspection_started_at  | None                                                          |
| instance_info          | {u'root_gb': u'200', u'display_name': u'bare6',               |
|                        | u'image_source': u'3601e669-f11b-46c4-8714-4895be15f59e',     |
|                        | u'memory_mb': u'32072', u'traits': [u'CUSTOM_TRAIT1'],        |
|                        | u'vcpus': u'8', u'local_gb': u'200', u'swap_mb': u'0',        |
|                        | u'nova_host_id': u'node004'}                                  |
| instance_uuid          | 5183bec6-112c-482c-bfe7-3a9f2fb97af5                          |
| last_error             | None                                                          |
| maintenance            | False                                                         |
| maintenance_reason     | None                                                          |
| management_interface   | ipmitool                                                      |
| name                   | node7                                                         |
| network_interface      | neutron                                                       |
| power_interface        | ipmitool                                                      |
| power_state            | power on                                                      |
```

```
| properties           | {u'memory_mb': 32072, u'cpu_arch': u'x86_64', u'local_gb':    |
|                      | 200, u'cpus': 8}                                              |
| provision_state      | active                                                        |
| provision_updated_at | 2019-01-14T14:03:13+00:00                                     |
| raid_config          | {}                                                            |
| raid_interface       | no-raid                                                       |
| rescue_interface     | no-rescue                                                     |
| reservation          | None                                                          |
| resource_class       | None                                                          |
| storage_interface    | noop                                                          |
| target_power_state   | None                                                          |
| target_provision_state | None                                                        |
| target_raid_config   | {}                                                            |
| traits               | [u'CUSTOM_TRAIT1']                                            |
| updated_at           | 2019-01-14T14:03:13+00:00                                     |
| uuid                 | bfedf771-6824-47b7-aa52-a3836ad20eb7                          |
| vendor_interface     | ipmitool                                                      |
+----------------------+---------------------------------------------------------------+
```

Troubleshooting

For troubleshooting, refer to the upstream docs:

https://docs.openstack.org/ironic/rocky/install/troubleshooting.html

ironic.conf

Below is the final ironic.conf file after applying all modifications:

```
[DEFAULT]

auth_strategy = keystone

enabled_hardware_types = ipmi,redfish

enabled_bios_interfaces = no-bios

enabled_boot_interfaces = pxe

enabled_deploy_interfaces = iscsi

enabled_inspect_interfaces = no-inspect,inspector

enabled_management_interfaces = ipmitool,redfish

enabled_power_interfaces = ipmitool,redfish

enabled_raid_interfaces = agent,no-raid

enabled_storage_interfaces = cinder,noop

enabled_vendor_interfaces = ipmitool,no-vendor

default_deploy_interface = iscsi

enabled_network_interfaces=noop,flat,neutron

default_network_interface=neutron

#this is the controller IP on the controller plane

my_ip=10.141.0.2

#use the RabbitMQ password for openstack user in rabbitmq same as other openstack services

transport_url=rabbit://openstack:nYrY6u4C5lE2mLuY9MJeVvwbMOsnkP@node002:5672/

[agent]

[ansible]

[api]

[audit]
```

[cimc]

[cinder]

[cisco_ucs]

[conductor]

automated_clean = false

clean_callback_timeout = 300

[console]

[cors]

[database]

#DataBase credentials you created before

connection=mysql+pymysql://ironic:system@10.141.255.245:3308/ironic?charset=utf8

[deploy]

enable_ata_secure_erase = false

erase_devices_priority=0

continue_if_disk_secure_erase_fails = true

default_boot_option = local

[dhcp]

[disk_partitioner]

[disk_utils]

[drac]

[glance]

auth_type = password

auth_url=http://10.141.255.245:5000/

username=ironic

password=system

project_name=service

project_domain_id=82f3bdbf990f4f8ba40100f25d034079

user_domain_id=82f3bdbf990f4f8ba40100f25d034079

region_name = openstack

valid_interfaces=public

[healthcheck]

[ilo]

[inspector]

auth_type = password

auth_url=http://10.141.255.245:5000/

username=ironic

password=system

project_name=service

project_domain_id=82f3bdbf990f4f8ba40100f25d034079

user_domain_id=82f3bdbf990f4f8ba40100f25d034079

region_name = openstack

valid_interfaces=public

[ipmi]

[irmc]

[ironic_lib]

[iscsi]

[keystone_authtoken]

auth_type=password

www_authenticate_uri=http://osdev.brightcomputing.com:5000

auth_url=http://10.141.255.245:5000

username=ironic

password=system

project_name=service

project_domain_name=Default

user_domain_name=Default

[matchmaker_redis]

[metrics]

[metrics_statsd]

[neutron]

auth_type = password

auth_url=http://10.141.255.245:5000/

username=ironic

password=system

project_name=service

#Replace the default domain ID with yours

project_domain_id=82f3bdbf990f4f8ba40100f25d034079

user_domain_id=82f3bdbf990f4f8ba40100f25d034079

region_name = openstack

valid_interfaces=public

#this is the network you will use for cleaning and provisioning, in our case we are using the bright-internal-flat-internalnet

cleaning_network = 5590588b-900c-407c-8a90-efb3e9694746

provisioning_network = 5590588b-900c-407c-8a90-efb3e9694746

[oneview]

[oslo_concurrency]

[oslo_messaging_amqp]

[oslo_messaging_kafka]

[oslo_messaging_notifications]

[oslo_messaging_rabbit]

[oslo_messaging_zmq]

[oslo_policy]

[profiler]

[pxe]

[service_catalog]

auth_type = password

auth_url=http://10.141.255.245:5000/

username=ironic

password=system

project_name=service

project_domain_id=82f3bdbf990f4f8ba40100f25d034079

user_domain_id=82f3bdbf990f4f8ba40100f25d034079

region_name = openstack

valid_interfaces=public

[snmp]

[ssl]

[swift]

auth_type = password

auth_url=http://10.141.255.245:5000/

username=ironic

password=system

project_name=service

project_domain_id=82f3bdbf990f4f8ba40100f25d034079

user_domain_id=82f3bdbf990f4f8ba40100f25d034079

region_name = openstack

valid_interfaces=public

[xclarity]

Unique solution ID: #1448
Author: Frank Furter
Last update: 2019-02-06 11:46