Red Hat OpenStack Platform 13

Bare Metal Provisioning

Install, Configure, and Use the Bare Metal Service (Ironic)

Last Updated: 2021-04-26

Red Hat OpenStack Platform 13 Bare Metal Provisioning

Install, Configure, and Use the Bare Metal Service (Ironic)

OpenStack Team
rhos-docs@redhat.com

Abstract

This guide provides procedures for installing, configuring, and using the Bare Metal service in the overcloud of a Red Hat OpenStack Platform environment.

Table of Contents

PREFACE

This document provides instructions for installing and configuring the Bare Metal service (ironic) in the overcloud, and using the service to provision and manage physical machines for end users.

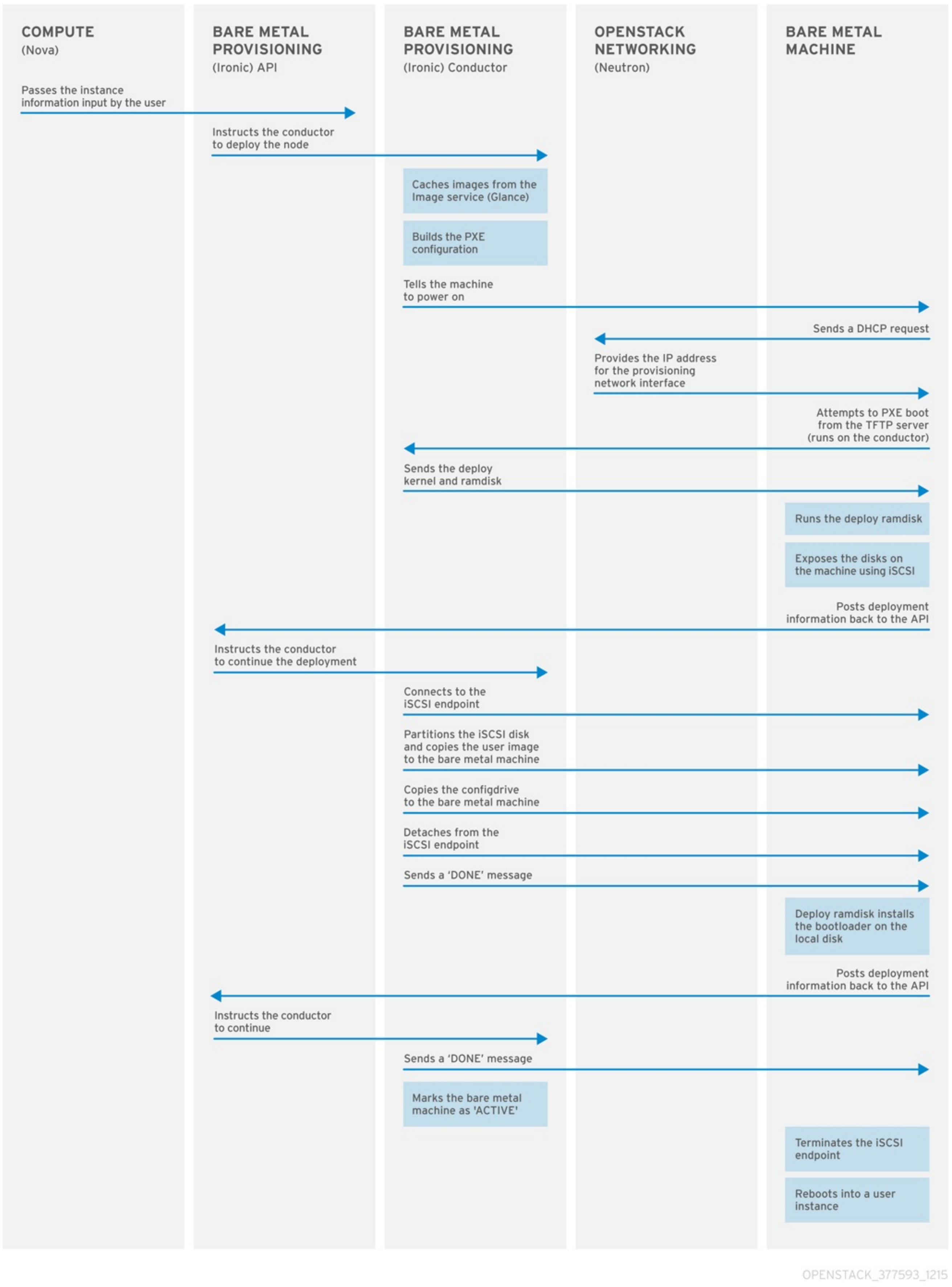
The Bare Metal service components are also used by the Red Hat OpenStack Platform director, as part of the undercloud, to provision and manage the bare metal nodes that make up the OpenStack environment (the overcloud). For information on how the director uses the Bare Metal service, see Director Installation and Usage.

CHAPTER 1. ABOUT THE BARE METAL SERVICE

The OpenStack Bare Metal service (ironic) provides the components required to provision and manage physical machines for end users. The Bare Metal service in the overcloud interacts with the following OpenStack services:

- OpenStack Compute (nova) provides scheduling, tenant quotas, IP assignment, and a user-facing API for virtual machine instance management, while the Bare Metal service provides the administrative API for hardware management.
- OpenStack Identity (keystone) provides request authentication and assists the Bare Metal service in locating other OpenStack services.
- OpenStack Image service (glance) manages images and image metadata.
- OpenStack Networking (neutron) provides DHCP and network configuration.
- OpenStack Object Storage (swift) is used by certain drivers to expose temporary URLs to images.

The Bare Metal service uses iPXE to provision physical machines. The following diagram outlines how the OpenStack services interact during the provisioning process when a user launches a new machine with the default drivers.


COMPUTE (Nova)
BARE METAL PROVISIONING (Ironic) API
BARE METAL PROVISIONING (Ironic) Conductor
OPENSTACK NETWORKING (Neutron)
BARE METAL MACHINE
Passes the instance information input by the user
Instructs the conductor to deploy the node
Caches images from the Image service (Glance)
Builds the PXE configuration
Tells the machine to power on
Sends a DHCP request
Provides the IP address for the provisioning network interface
Attempts to PXE boot from the TFTP server (runs on the conductor)
Sends the deploy kernel and ramdisk
Runs the deploy ramdisk
Exposes the disks on the machine using iSCSI
Posts deployment information back to the API
Instructs the conductor to continue the deployment
Connects to the iSCSI endpoint
Partitions the iSCSI disk and copies the user image to the bare metal machine
Copies the configdrive to the bare metal machine
Detaches from the iSCSI endpoint
Sends a 'DONE' message
Deploy ramdisk installs the bootloader on the local disk
Posts deployment information back to the API
Instructs the conductor to continue
Sends a 'DONE' message
Marks the bare metal machine as 'ACTIVE'
Terminates the iSCSI endpoint
Reboots into a user instance
OPENSTACK_377593_1215

CHAPTER 2. PLANNING FOR BARE METAL PROVISIONING

This chapter outlines the requirements for setting up the Bare Metal service, including installation assumptions, hardware requirements, and networking requirements.

2.1. INSTALLATION ASSUMPTIONS

This guide assumes you have installed the director on the undercloud node, and are ready to install the Bare Metal service along with the rest of the overcloud. For more information on installing the director, see Installing the Undercloud.

NOTE

The Bare Metal service in the overcloud is designed for a trusted tenant environment, as the bare metal nodes have direct access the control plane network of your OpenStack installation. If you implement a custom composable network for Ironic services in the overcloud, users do not need to access the control plane.

2.2. HARDWARE REQUIREMENTS

Overcloud Requirements

The hardware requirements for an overcloud with the Bare Metal service are the same as for the standard overcloud. For more information, see Overcloud Requirements in the *Director Installation and Usage* guide.

Bare Metal Machine Requirements

The hardware requirements for bare metal machines that will be provisioned vary depending on the operating system you are installing. For Red Hat Enterprise Linux 7, see the Red Hat Enterprise Linux 7 Installation Guide. For Red Hat Enterprise Linux 6, see the Red Hat Enterprise Linux 6 Installation Guide .

All bare metal machines to be provisioned require the following:

- A NIC to connect to the bare metal network.
- A power management interface (for example, IPMI) connected to a network reachable from the **ironic-conductor** service. By default, **ironic-conductor** runs on all of the controller nodes, unless you are using composable roles and running **ironic-conductor** elsewhere.
- PXE boot on the bare metal network. Disable PXE boot on all other NICs in the deployment.

2.3. NETWORKING REQUIREMENTS

NOTE

If you use OVN, the Bare Metal Provisioning service (ironic) is not supported in the overcloud. The built-in DHCP server in OVN presently cannot provision bare metal nodes or serve DHCP for the provisioning networks.

The bare metal network:

This is a private network that the Bare Metal service uses for:

- The provisioning and management of bare metal machines on the overcloud.
- Cleaning bare metal nodes before and between deployments.
- Tenant access to the bare metal nodes.

The bare metal network provides DHCP and PXE boot functions to discover bare metal systems. This network must use a native VLAN on a trunked interface so that the Bare Metal service can serve PXE boot and DHCP requests.

You can configure the bare metal network in two ways:

- Use a flat bare metal network for Ironic Conductor services. This network must route to the Ironic services on the control plane. If you define an isolated bare metal network, the bare metal notes cannot PXE boot.
- Use a custom composable network to implement Ironic services in the overcloud.

NOTE

The Bare Metal service in the overcloud is designed for a trusted tenant environment, as the bare metal nodes have direct access to the control plane network of your OpenStack installation. If you implement a custom composable network for Ironic services in the overcloud, users do not need to access the control plane.

Network tagging:

- The control plane network (the director’s provisioning network) is always untagged.
- The bare metal network must be untagged for provisioning, and must also have access to the Ironic API.
- Other networks may be tagged.

Overcloud controllers:

The controller nodes with the Bare Metal service must have access to the bare metal network.

Bare metal nodes:

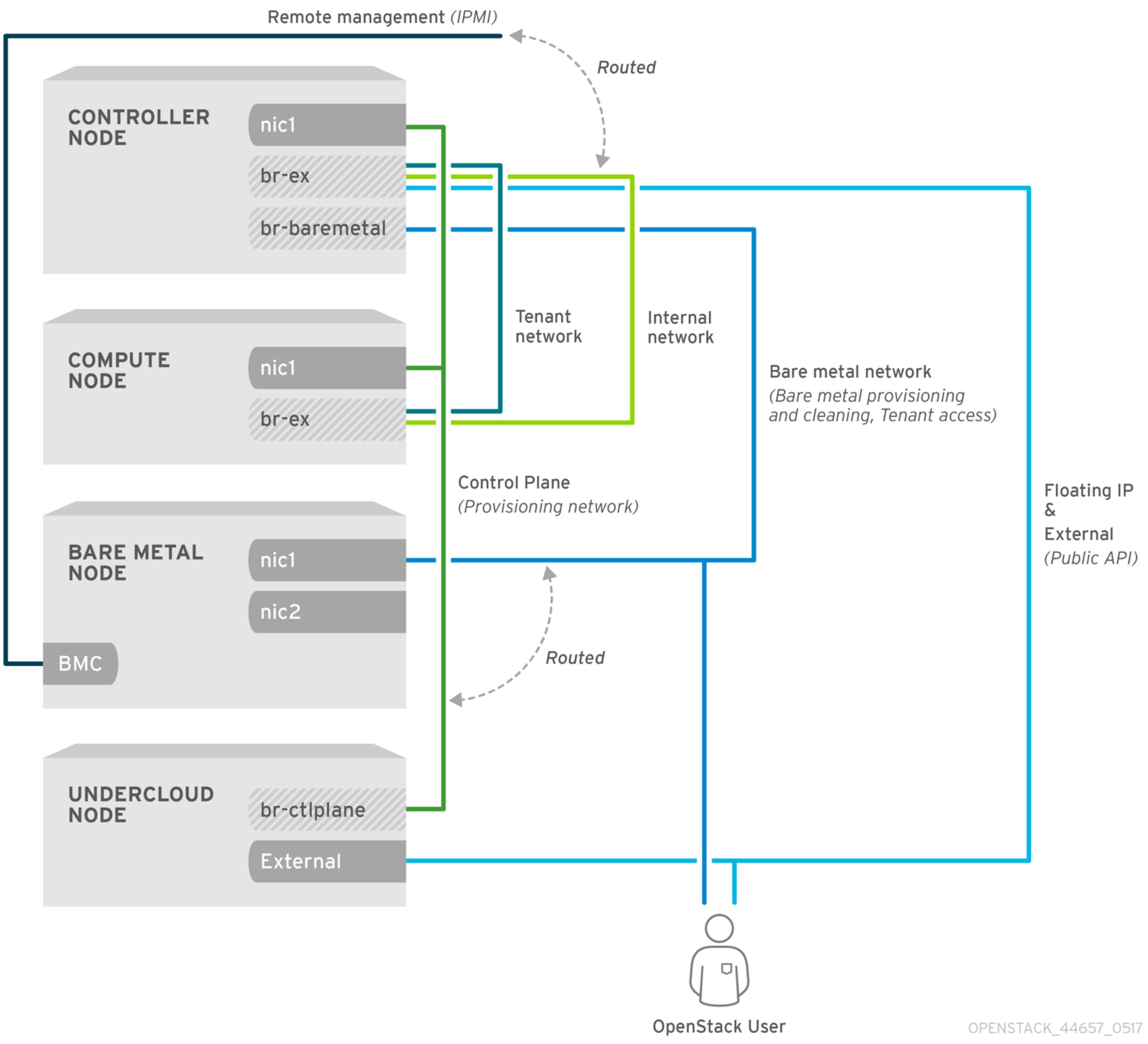
The NIC which the bare metal node is configured to PXE-boot from must have access to the bare metal network.

2.3.1. The Default Bare Metal Network

In this architecture, the bare metal network is separated from the control plane network. The bare metal network is a flat network that also acts as the tenant network.

- The bare metal network is created by the OpenStack operator. This network requires a route to the director’s provisioning network.
- Ironic users have access to the public OpenStack APIs, and to the bare metal network. Since the bare metal network is routed to the director’s provisioning network, users also have indirect access to the control plane.
- Ironic uses the bare metal network for node cleaning.

Default bare metal network architecture diagram



2.3.2. The Custom Composable Network

In this architecture, the bare metal network is a custom composable network that does not have access to the control plane. Creating this network may be preferable if you want to limit access to the control plane.

- The custom composable bare metal network is created by the OpenStack operator.
- Ironic users have access to the public OpenStack APIs, and to the custom composable bare metal network.
- Ironic uses the custom composable bare metal network for node cleaning.

CHAPTER 3. DEPLOYING AN OVERCLOUD WITH THE BARE METAL SERVICE

For full details about overcloud deployment with the director, see Director Installation and Usage. This chapter only covers deployment steps specific to ironic.

3.1. CREATING THE IRONIC TEMPLATE

Use an environment file to deploy the overcloud with the Bare Metal service enabled. A template is located on the director node at **/usr/share/openstack-tripleo-heat-templates/environments/services-docker/ironic.yaml**.

Filling in the template

Additional configuration can be specified either in the provided template or in an additional yaml file, for example **~/templates/ironic.yaml**.

- For a hybrid deployment with both bare metal and virtual instances, you must add **AggregateInstanceExtraSpecsFilter** to the list of **NovaSchedulerDefaultFilters**. If you have not set **NovaSchedulerDefaultFilters** anywhere, you can do so in ironic.yaml. For an example, see Section 3.3, "Example Templates".

  NOTE

  If you are using SR-IOV, NovaSchedulerDefaultFilters is already set in **tripleo-heat-templates/environments/neutron-sriov.yaml**. Append **AggregateInstanceExtraSpecsFilter** to this list.

- The type of cleaning that occurs before and between deployments is set by **IronicCleaningDiskErase**. By default, this is set to 'full' by **puppet/services/ironic-conductor.yaml**. Setting this to 'metadata' can substantially speed up the process, as it only cleans the partition table, however, since the deployment will be less secure in a multi-tenant environment, you should only do this in a trusted tenant environment.

- You can add drivers with the **IronicEnabledDrivers** parameter. By default, **pxe_ipmitool**, **pxe_drac** and **pxe_ilo** are enabled.

For a full list of configuration parameters, see Bare Metal in the *Overcloud Parameters* guide.

3.2. NETWORK CONFIGURATION

If you use the default flat bare metal network, you must create a bridge **br-baremetal** for ironic to use. You can specify this in an additional template:

~/templates/network-environment.yaml

```
parameter_defaults:
  NeutronBridgeMappings: datacentre:br-ex,baremetal:br-baremetal
  NeutronFlatNetworks: datacentre,baremetal
```

You can either configure this bridge in the provisioning network (control plane) of the controllers, so you can reuse this network as the bare metal network, or add a dedicated network. The configuration requirements are the same, however the bare metal network cannot be VLAN-tagged, as it is used for provisioning.

~/templates/nic-configs/controller.yaml

```
network_config:
   -
     type: ovs_bridge
         name: br-baremetal
         use_dhcp: false
         members:
           -
             type: interface
             name: eth1
```

NOTE

The Bare Metal service in the overcloud is designed for a trusted tenant environment, as the bare metal nodes have direct access to the control plane network of your OpenStack installation.

3.2.1. Configuring a Custom Provisioning Network

The default flat provisioning network can introduce security concerns in a customer environment as a tenant can interfere with the undercloud network. To prevent this risk, you can configure a custom composable bare metal provisioning network for ironic services that does not have access to the control plane:

1. Configure the shell to access Identity as the administrative user:

   ```
   $ source ~/overcloudrc
   ```

2. Copy the **network_data.yaml** file:

   ```
   (undercloud) [stack@host01 ~]$ cp /usr/share/openstack-tripleo-heat-
   templates/network_data.yaml .
   ```

3. Edit the new **network_data.yaml** file and add a new network for Overcloud provisioning:

   ```
   # custom network for Overcloud provisioning
   - name: OcProvisioning
   name_lower: oc_provisioning
   vip: true
   vlan: 205
   ip_subnet: '172.23.3.0/24'
   allocation_pools: [{'start': '172.23.3.10', 'end': '172.23.3.200'}]
   ```

4. Update the **network_environments.yaml** and **nic-configs/controller.yaml** files to use the new network.

   a. In the **network_environments.yaml** file, add Vlan and remap Ironic networks:

      ```
      ServiceNetMap:
        IronicApiNetwork: oc_provisioning
        IronicNetwork: oc_provisioning
      ```

b. In the **nic-configs/controller.yaml** file, add an interface and necessary parameters:

```
$network_config:
     - type: vlan
        vlan_id:
          get_param: OcProvisioningNetworkVlanID
        addresses:
        - ip_netmask:
            get_param: OcProvisioningIpSubnet
```

5. Copy the **roles_data.yaml** file:

```
(undercloud) [stack@host01 ~]$ cp /usr/share/openstack-tripleo-heat-
templates/roles_data.yaml .
```

6. Edit the new **roles_data.yaml** and add the new network for the controller:

```
  networks:
   ...
    - OcProvisioning
```

7. Include the new **network_data.yaml** and **roles_data.yaml** files in the deploy command:

```
-n /home/stack/network_data.yaml \
-r /home/stack/roles_data.yaml \
```

3.3. EXAMPLE TEMPLATES

The following is an example template file. This file may not meet the requirements of your environment. Before using this example, make sure it does not interfere with any existing configuration in your environment.

~/templates/ironic.yaml

```
parameter_defaults:

    NovaSchedulerDefaultFilters:
        - RetryFilter
        - AggregateInstanceExtraSpecsFilter
        - AvailabilityZoneFilter
        - RamFilter
        - DiskFilter
        - ComputeFilter
        - ComputeCapabilitiesFilter
        - ImagePropertiesFilter

    IronicCleaningDiskErase: metadata
```

In this example:

- The **AggregateInstanceExtraSpecsFilter** allows both virtual and bare metal instances, for a hybrid deployment.

- Disk cleaning that is done before and between deployments only erases the partition table (metadata).

3.4. DEPLOYING THE OVERCLOUD

To enable the Bare Metal service, include your ironic environment files with **-e** when deploying or redeploying the overcloud, along with the rest of your overcloud configuration.

For example:

```
$ openstack overcloud deploy \
  --templates \
  -e ~/templates/node-info.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml \
  -e ~/templates/network-environment.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/services-docker/ironic.yaml \
  -e ~/templates/ironic.yaml \
```

For more information about deploying the overcloud, see Creating the Overcloud with the CLI Tools and Including Environment Files in Overcloud Creation .

3.5. TESTING THE BARE METAL SERVICE

You can use the OpenStack Integration Test Suite to validate your Red Hat OpenStack deployment. For more information, see the OpenStack Integration Test Suite Guide .

Additional Ways to Verify the Bare Metal Service:

1. Set up the shell to access Identity as the administrative user:

   ```
   $ source ~/overcloudrc
   ```

2. Check that the **nova-compute** service is running on the controller nodes:

   ```
   $ openstack compute service list -c Binary -c Host -c Status
   ```

3. If you have changed the default ironic drivers, make sure the required drivers are enabled:

   ```
   $ openstack baremetal driver list
   ```

4. Ensure that the ironic endpoints are listed:

   ```
   $ openstack catalog list
   ```

CHAPTER 4. CONFIGURING FOR THE BARE METAL SERVICE AFTER DEPLOYMENT

This section describes the steps necessary to configure your overcloud after deployment.

4.1. CONFIGURING OPENSTACK NETWORKING

Configure OpenStack Networking to communicate with the Bare Metal service for DHCP, PXE boot, and other requirements. You can configure the bare metal network in two ways:

- Use a flat bare metal network for Ironic Conductor services. This network must route to the Ironic services on the control plane network.
- Use a custom composable network to implement Ironic services in the overcloud.

Follow the procedures in this section to configure OpenStack Networking for a single flat network for provisioning onto bare metal, or to configure a new composable network that does not rely on an unused isolated network or a flat network. The configuration uses the ML2 plug-in and the Open vSwitch agent.

Perform all steps in the following procedure on the server that hosts the OpenStack Networking service, while logged in as the **root** user.

4.1.1. Configuring OpenStack Networking to Communicate with the Bare Metal Service on a flat Bare Metal Network

1. Configure the shell to access Identity as the administrative user:

```
$ source ~/overcloudrc
```

2. Create the flat network over which to provision bare metal instances:

```
$ openstack network create \
  --provider-network-type flat \
  --provider-physical-network baremetal \
  --share NETWORK_NAME
```

Replace *NETWORK_NAME* with a name for this network. To avoid erroneous reconfiguration in when you perform node cleaning, set this value to **provisioning**. The name of the physical network over which the virtual network is implemented, (in this case **baremetal**), was set earlier in **~/templates/network-environment.yaml**, with the parameter **NeutronBridgeMappings**.

3. Create the subnet on the flat network:

```
$ openstack subnet create \
  --network NETWORK_NAME \
  --subnet-range NETWORK_CIDR \
  --ip-version 4 \
  --gateway GATEWAY_IP \
  --allocation-pool start=START_IP,end=END_IP \
  --dhcp SUBNET_NAME
```

Replace the following values:

- Replace *SUBNET_NAME* with a name for the subnet.

- Replace *NETWORK_NAME* with the name of the provisioning network you created in the previous step.
- Replace *NETWORK_CIDR* with the Classless Inter-Domain Routing (CIDR) representation of the block of IP addresses the subnet represents. The block of IP addresses specified by the range started by *START_IP* and ended by END_IP must fall within the block of IP addresses specified by NETWORK_CIDR.
- Replace *GATEWAY_IP* with the IP address or host name of the router interface that will act as the gateway for the new subnet. This address must be within the block of IP addresses specified by *NETWORK_CIDR*, but outside of the block of IP addresses specified by the range started by *START_IP* and ended by *END_IP*.
- Replace *START_IP* with the IP address that denotes the start of the range of IP addresses within the new subnet from which floating IP addresses will be allocated.
- Replace *END_IP* with the IP address that denotes the end of the range of IP addresses within the new subnet from which floating IP addresses will be allocated.

4. Create a router for the network and subnet to ensure that the OpenStack Networking Service serves metadata requests:

   ```
   $ openstack router create ROUTER_NAME
   ```

   Replace **ROUTER_NAME** with a name for the router.

5. Attach the network to the new router:

   ```
   $ openstack router add network ROUTER_NAME NETWORK
   ```

   Replace *ROUTER_NAME* with the name of your router, and replace *NETWORK* with the ID or name of the network that you created previously.

6. Attach the subnet to the new router:

   ```
   $ openstack router add subnet ROUTER_NAME BAREMETAL_SUBNET
   ```

   Replace *ROUTER_NAME* with the name of your router and *BAREMETAL_SUBNET* with the ID or name of the subnet that you created previously. This allows the metadata requests from **cloud-init** to be served and the node configured.

4.1.2. Configuring OpenStack Networking to Communicate with the Bare Metal Service on a Custom Composable Bare Metal Network

1. Create a vlan network with a VlanID that matches the **OcProvisioning** network that you create during deployment. Name the new network **provisioning** to match the default name of the cleaning network.

   ```
   (overcloud) [stack@host01 ~]$ openstack network create \
     --share \
     --provider-network-type vlan \
     --provider-physical-network datacentre \
     --provider-segment 205 provisioning
   ```

If the name of the overcloud network is not **provisioning**, log onto the controller and run the following commands to rename and restart the network:

```
heat-admin@overcloud-controller-0 ~]$ sudo vi /var/lib/config-data/puppet-
generated/ironic/etc/ironic/ironic.conf
```

```
heat-admin@overcloud-controller-0 ~]$ sudo docker restart ironic_conductor
```

4.2. CONFIGURING NODE CLEANING

By default, the Bare Metal service is set to use a network named **provisioning** for node cleaning. However, network names are not unique in OpenStack Networking, so it is possible for a tenant to create a network with the same name, causing a conflict with the Bare Metal service. Therefore, it is recommended to use the network UUID instead.

1. Configure cleaning by providing the provider network UUID on the controller running the Bare Metal Service:
   ~/templates/ironic.yaml

   ```
   parameter_defaults:
     IronicCleaningNetwork: UUID
   ```

   Replace *UUID* with the UUID of the bare metal network created in the previous steps.

   You can find the UUID using **openstack network show**:

   ```
   openstack network show NETWORK_NAME -f value -c id
   ```

   NOTE

   This configuration must be done after the initial overcloud deployment, because the UUID for the network is not available beforehand.

2. Apply the changes by redeploying the overcloud with the **openstack overcloud deploy** command as described in Section 3.4, “Deploying the Overcloud”. Redeploying the overcloud with **openstack overcloud deploy** reverts any manual changes, so make sure you have added the cleaning configuration to **~/templates/ironic.yaml** (described in the previous step) before the next time you use **openstack overcloud deploy**.

3. You can also apply the change manually, without using director. Configure the cleaning network:

   ```
   crudini --set /var/lib/config-data/puppet-generated/ironic/etc/ironic/ironic.conf neutron
   cleaning_network
   ```

4. Restart the Bare Metal Provisioning service:

   ```
   # docker restart ironic_conductor
   ```

4.2.1. Manual Node Cleaning

To manually initiate node cleaning, the node must be in the **manageable** state.

Node cleaning has two modes:

Metadata only clean - Removes partitions from all disks on a given node. This is a faster clean cycle, but less secure since it only erases partition tables. Only use this mode on trusted tenant environments.

Full clean - Removes all data from all disks, using either ATA secure erase or by shredding. This can take several hours to complete.

To initiate a **metadata** clean:

```
$ openstack baremetal node clean _UUID_ \
    --clean-steps '[{"interface": "deploy", "step": "erase_devices_metadata"}]'
```

To initiate a **full** clean:

```
$ openstack baremetal node clean _UUID_ \
    --clean-steps '[{"interface": "deploy", "step": "erase_devices"}]'
```

Replace *UUID* with the UUID of the node you would like cleaned.

After a successful cleaning, the node state returns to **manageable**. If the state is **clean failed**, check the **last_error** field for the cause of failure.

4.3. CREATING THE BARE METAL FLAVOR

You need to create a flavor to use as a part of the deployment. The specifications (memory, CPU, and disk) of this flavor must be equal to or less than what your bare metal node provides.

1. Set up the shell to access Identity as the administrative user:

   ```
   $ source ~/overcloudrc
   ```

2. List existing flavors:

   ```
   $ openstack flavor list
   ```

3. Create a new flavor for the Bare Metal service:

   ```
   $ openstack flavor create \
     --id auto --ram RAM \
     --vcpus VCPU --disk DISK \
     --property baremetal=true \
     --public baremetal
   ```

   Replace **RAM** with the amount of memory, **VCPU** with the number of vCPUs and **DISK** with the disk storage value. The property **baremetal** is used to distinguish bare metal from virtual instances.

4. Verify that the new flavor is created with the respective values:

   ```
   $ openstack flavor list
   ```

4.4. CREATING THE BARE METAL IMAGES

The deployment requires two sets of images:

- The **deploy image** is used by the Bare Metal service to boot the bare metal node and copy a user image onto the bare metal node. The deploy image consists of the **kernel** image and the **ramdisk** image.
- The **user image** is the image deployed onto the bare metal node. The user image also has a **kernel** image and **ramdisk** image, but additionally, the user image contains a **main** image. The main image is either a root partition, or a whole-disk image.
  - A **whole-disk image** is an image that contains the partition table and boot loader. The Bare Metal service does not control the subsequent reboot of a node deployed with a whole-disk image as the node supports localboot.
  - A **root partition image** only contains the root partition of the operating system. If using a root partition, after the deploy image is loaded into the Image service, you can set the deploy image as the node’s boot image in the node’s properties. A subsequent reboot of the node uses netboot to pull down the user image.

The examples in this section use a root partition image to provision bare metal nodes.

4.4.1. Preparing the Deploy Images

You do not have to create the deploy image as it was already used when the overcloud was deployed by the undercloud. The deploy image consists of two images - the kernel image and the ramdisk image as follows:

```
/tftpboot/agent.kernel
/tftpboot/agent.ramdisk
```

These images are often in the home directory, unless you have deleted them, or unpacked them elsewhere. If they are not in the home directory, and you still have the **rhosp-director-images-ipa** package installed, these images will be in the **/usr/share/rhosp-director-images/ironic-python-agent*.tar** file.

Extract the images and upload them to the Image service:

```
$ mkdir images
$ tar -xf /usr/share/rhosp-director-images/ironic-python-agent-latest.tar -C images/
$ cd images
$ openstack image create \
  --container-format aki \
  --disk-format aki \
  --public \
  --file ./tftpboot/agent.kernel bm-deploy-kernel
$ openstack image create \
  --container-format ari \
  --disk-format ari \
  --public \
  --file ./tftpboot/agent.ramdisk bm-deploy-ramdisk
```

4.4.2. Preparing the User Image

The final image that you need is the user image that will be deployed on the bare metal node. User images also have a kernel and ramdisk, along with a main image. To download and install these packages, you must first configure whole disk image environment variables to suit your requirements.

4.4.2.1. Disk Image Environment Variables

As a part of the disk image building process, the director requires a base image and registration details to obtain packages for the new overcloud image. You define these aspects using Linux environment variables.

NOTE

The image building process temporarily registers the image with a Red Hat subscription and unregisters the system once the image building process completes.

To build a disk image, set Linux environment variables that suit your environment and requirements:

DIB_LOCAL_IMAGE

Sets the local image to use as your basis.

REG_ACTIVATION_KEY

Use an activation key instead as part of the registration process.

REG_AUTO_ATTACH

Defines whether or not to automatically attach the most compatible subscription.

REG_BASE_URL

The base URL of the content delivery server to pull packages. The default Customer Portal Subscription Management process uses **https://cdn.redhat.com**. If using a Red Hat Satellite 6 server, this parameter should use the base URL of your Satellite server.

REG_ENVIRONMENT

Registers to an environment within an organization.

REG_METHOD

Sets the method of registration. Use **portal** to register a system to the Red Hat Customer Portal. Use **satellite** to register a system with Red Hat Satellite 6.

REG_ORG

The organization to register the images.

REG_POOL_ID

The pool ID of the product subscription information.

REG_PASSWORD

Gives the password for the user account registering the image.

REG_REPOS

A string of repository names separated with commas (no spaces). Each repository in this string is enabled through **subscription-manager**.

REG_SAT_URL

The base URL of the Satellite server to register Overcloud nodes. Use the Satellite's HTTP URL and not the HTTPS URL for this parameter. For example, use http://satellite.example.com and not https://satellite.example.com.

REG_SERVER_URL

Gives the hostname of the subscription service to use. The default is for the Red Hat Customer Portal at **subscription.rhn.redhat.com**. If using a Red Hat Satellite 6 server, this parameter should use the hostname of your Satellite server.

REG_USER

Gives the user name for the account registering the image.

4.4.3. Installing the User Image

1. Download the Red Hat Enterprise Linux KVM guest image from the Customer Portal (requires login).

2. Define DIB_LOCAL_IMAGE as the downloaded image:

```
$ export DIB_LOCAL_IMAGE=rhel-server-7.4-x86_64-kvm.qcow2
```

3. Set your registration information. If you use Red Hat Customer Portal, you must configure the following information:

```
$ export REG_USER='USER_NAME'
$ export REG_PASSWORD='PASSWORD'
$ export REG_AUTO_ATTACH=true
$ export REG_METHOD=portal
$ export https_proxy='IP_address:port' (if applicable)
$ export http_proxy='IP_address:port' (if applicable)
```

If you use Red Hat Satellite, you must configure the following information:

```
$ export REG_USER='USER_NAME'
$ export REG_PASSWORD='PASSWORD'
$ export REG_SAT_URL='<SATELLITE URL>'
$ export REG_ORG='<SATELLITE ORG>'
$ export REG_ENV='<SATELLITE ENV>'
$ export REG_METHOD=<METHOD>
```

If you have any offline repositories, you can define DIB_YUM_REPO_CONF as local repository configuration:

```
$ export DIB_YUM_REPO_CONF=<path-to-local-repository-config-file>
```

4. Create the user images using the **diskimage-builder** tool:

```
$ disk-image-create rhel7 baremetal -o rhel-image
```

This extracts the kernel as **rhel-image.vmlinuz** and initial ramdisk as **rhel-image.initrd**.

5. Upload the images to the Image service:

```
$ KERNEL_ID=$(openstack image create \
  --file rhel-image.vmlinuz --public \
  --container-format aki --disk-format aki \
  -f value -c id rhel-image.vmlinuz)
$ RAMDISK_ID=$(openstack image create \
  --file rhel-image.initrd --public \
  --container-format ari --disk-format ari \
  -f value -c id rhel-image.initrd)
$ openstack image create \
  --file rhel-image.qcow2   --public \
  --container-format bare \
  --disk-format qcow2 \
  --property kernel_id=$KERNEL_ID \
  --property ramdisk_id=$RAMDISK_ID \
  rhel-image
```

4.5. ADDING PHYSICAL MACHINES AS BARE METAL NODES

There are two methods to enroll a bare metal node:

1. Prepare an inventory file with the node details, import the file into the Bare Metal service, then make the nodes available.

2. Register a physical machine as a bare metal node, then manually add its hardware details and create ports for each of its Ethernet MAC addresses. These steps can be performed on any node which has your overcloudrc file.

Both methods are detailed in this section.

After enrolling the physical machines, Compute is not immediately notified of new resources, because Compute's resource tracker synchronizes periodically. Changes will be visible after the next periodic task is run. This value, **scheduler_driver_task_period**, can be updated in ***/etc/nova/nova.conf***. The default period is 60 seconds.

4.5.1. Enrolling a Bare Metal Node With an Inventory File

1. Create a file **overcloud-nodes.yaml**, including the node details. Multiple nodes can be enrolled with one file.

```
nodes:
    - name: node0
      driver: pxe_ipmitool
      driver_info:
        ipmi_address: <IPMI_IP>
        ipmi_username: <USER>
        ipmi_password: <PASSWORD>
      properties:
        cpus: <CPU_COUNT>
        cpu_arch: <CPU_ARCHITECTURE>
        memory_mb: <MEMORY>
        local_gb: <ROOT_DISK>
        root_device:
            serial: <SERIAL>
      ports:
        - address: <PXE_NIC_MAC>
```

Replace the following values:

- **<IPMI_IP>** with the address of the Bare Metal controller.

- **<USER>** with your username.

- **<PASSWORD>** with your password.
- **<CPU_COUNT>** with the number of CPUs.
- **<CPU_ARCHITECTURE>** with the type of architecture of the CPUs.
- **<MEMORY>** with the amount of memory in MiB.
- **<ROOT_DISK>** with the size of the root disk in GiB.
- **<PXE_NIC_MAC>** with the MAC address of the NIC used to PXE boot.
  You only need to include **root_device** if the machine has multiple disks. Replace **<SERIAL>** with the serial number of the disk you would like used for deployment.

2. Set up the shell to use Identity as the administrative user:

```
$ source ~/overcloudrc
```

3. Import the inventory file into ironic:

```
$ openstack baremetal create overcloud-nodes.yaml
```

4. The nodes are now in the **enroll** state.

5. Specify the deploy kernel and deploy ramdisk on each node:

```
$ openstack baremetal node set NODE_UUID \
  --driver-info deploy_kernel=KERNEL_UUID \
  --driver-info deploy_ramdisk=INITRAMFS_UUID
```

   Replace the following values:

   - Replace *NODE_UUID* with the unique identifier for the node. Alternatively, use the node's logical name.
   - Replace *KERNEL_UUID* with the unique identifier for the **kernel** deploy image that was uploaded to the Image service. Find this value with:

```
$ openstack image show bm-deploy-kernel -f value -c id
```

   - Replace *INITRAMFS_UUID* with the unique identifier for the **ramdisk** image that was uploaded to the Image service. Find this value with:

```
$ openstack image show bm-deploy-ramdisk -f value -c id
```

6. Set the node's provisioning state to **available**:

```
$ openstack baremetal node manage NODE_UUID
$ openstack baremetal node provide NODE_UUID
```

   The bare metal service cleans the node if you enabled node cleaning,

7. Set the local boot option on the node:

```
$ openstack baremetal node set _NODE_UUID_ --property capabilities="boot_option:local"
```

8. Check that the nodes were successfully enrolled:

```
$ openstack baremetal node list
```

There may be a delay between enrolling a node and its state being shown.

4.5.2. Enrolling a Bare Metal Node Manually

1. Set up the shell to use Identity as the administrative user:

```
$ source ~/overcloudrc
```

2. Add a new node:

```
$ openstack baremetal node create --driver pxe_ipmitool --name NAME
```

To create a node you must specify the driver name. This example uses **pxe_impitool**. To use a different driver, you must enable it by setting the **IronicEnabledDrivers** parameter. For more information on supported drivers, see Appendix A, *Bare Metal Drivers*.

IMPORTANT

Note the unique identifier for the node.

3. Update the node driver information to allow the Bare Metal service to manage the node:

```
$ openstack baremetal node set NODE_UUID \
  --driver_info PROPERTY=VALUE \
  --driver_info PROPERTY=VALUE
```

Replace the following values:

- Replace *NODE_UUID* with the unique identifier for the node. Alternatively, use the node's logical name.
- Replace *PROPERTY* with a required property returned by the **ironic driver-properties** command.
- Replace *VALUE* with a valid value for that property.

4. Specify the deploy kernel and deploy ramdisk for the node driver:

```
$ openstack baremetal node set NODE_UUID \
  --driver-info deploy_kernel=KERNEL_UUID \
  --driver-info deploy_ramdisk=INITRAMFS_UUID
```

Replace the following values:

- Replace *NODE_UUID* with the unique identifier for the node. Alternatively, use the node's logical name.

- Replace *KERNEL_UUID* with the unique identifier for the ***.kernel*** image that was uploaded to the Image service.
- Replace *INITRAMFS_UUID* with the unique identifier for the ***.initramfs*** image that was uploaded to the Image service.

5. Update the node's properties to match the hardware specifications on the node:

```
$ openstack baremetal node set NODE_UUID \
  --property cpus=CPU \
  --property memory_mb=RAM_MB \
  --property local_gb=DISK_GB \
  --property cpu_arch=ARCH
```

   Replace the following values:

   - Replace *NODE_UUID* with the unique identifier for the node. Alternatively, use the node's logical name.
   - Replace *CPU* with the number of CPUs.
   - Replace *RAM_MB* with the RAM (in MB).
   - Replace *DISK_GB* with the disk size (in GB).
   - Replace *ARCH* with the architecture type.

6. OPTIONAL: Configure the node to reboot after initial deployment from a local boot loader installed on the node's disk, instead of using PXE from **ironic-conductor**. The local boot capability must also be set on the flavor used to provision the node. To enable local boot, the image used to deploy the node must contain **grub2**. Configure local boot:

```
$ openstack baremetal node set NODE_UUID \
  --property capabilities="boot_option:local"
```

   Replace *NODE_UUID* with the unique identifier for the node. Alternatively, use the node's logical name.

7. Inform the Bare Metal service of the node's network card by creating a port with the MAC address of the NIC on the provisioning network:

```
$ openstack baremetal port create --node NODE_UUID MAC_ADDRESS
```

   Replace *NODE_UUID* with the unique identifier for the node. Replace *MAC_ADDRESS* with the MAC address of the NIC used to PXE boot.

8. If you have multiple disks, set the root device hints. This informs the deploy ramdisk which disk it should use for deployment.

```
$ openstack baremetal node set NODE_UUID \
  --property root_device={"PROPERTY": "VALUE"}
```

   Replace with the following values:

   - Replace *NODE_UUID* with the unique identifier for the node. Alternatively, use the node's logical name.

- Replace PROPERTY and VALUE with details about the disk you want used for deployment, for example **root_device='{"size": 128}'**
  The following properties are supported:
  - **model** (String): Device identifier.
  - **vendor** (String): Device vendor.
  - **serial** (String): Disk serial number.
  - **hctl** (String): Host:Channel:Target:Lun for SCSI.
  - **size** (Integer): Size of the device in GB.
  - **wwn** (String): Unique storage identifier.
  - **wwn_with_extension** (String): Unique storage identifier with the vendor extension appended.
  - **wwn_vendor_extension** (String): Unique vendor storage identifier.
  - **rotational** (Boolean): True for a rotational device (HDD), otherwise false (SSD).
  - **name** (String): The name of the device, for example: /dev/sdb1 Only use this for devices with persistent names.

  NOTE

  If you specify more than one property, the device must match all of those properties.

9. Validate the node’s setup:

```
$ openstack baremetal node validate NODE_UUID
+------------+--------+---------------------------------------------+
| Interface  | Result | Reason                                      |
+------------+--------+---------------------------------------------+
| boot       | False  | Cannot validate image information for node  |
|            |        | a02178db-1550-4244-a2b7-d7035c743a9b        |
|            |        | because one or more parameters are missing  |
|            |        | from its instance_info. Missing are:        |
|            |        | ['ramdisk', 'kernel', 'image_source']       |
| console    | None   | not supported                               |
| deploy     | False  | Cannot validate image information for node  |
|            |        | a02178db-1550-4244-a2b7-d7035c743a9b        |
|            |        | because one or more parameters are missing  |
|            |        | from its instance_info. Missing are:        |
|            |        | ['ramdisk', 'kernel', 'image_source']       |
| inspect    | None   | not supported                               |
| management | True   |                                             |
| network    | True   |                                             |
| power      | True   |                                             |
| raid       | True   |                                             |
| storage    | True   |                                             |
+------------+--------+---------------------------------------------+
```

Replace *NODE_UUID* with the unique identifier for the node. Alternatively, use the node’s logical name. The output of the command above should report either **True** or **None** for each interface. Interfaces marked **None** are those that you have not configured, or those that are not supported for your driver.

NOTE

Interfaces may fail validation due to missing 'ramdisk', 'kernel', and 'image_source' parameters. This result is fine, because the Compute service populates those missing parameters at the beginning of the deployment process.

4.6. USING HOST AGGREGATES TO SEPARATE PHYSICAL AND VIRTUAL MACHINE PROVISIONING

OpenStack Compute uses host aggregates to partition availability zones, and group together nodes with specific shared properties. When an instance is provisioned, Compute’s scheduler compares properties on the flavor with the properties assigned to host aggregates, and ensures that the instance is provisioned in the correct aggregate and on the correct host: either on a physical machine or as a virtual machine.

The procedure below describes how to do the following:

- Add the property **baremetal** to your flavors, setting it to either **true** or **false**.
- Create separate host aggregates for bare metal hosts and compute nodes with a matching **baremetal** property. Nodes grouped into an aggregate inherit this property.

Creating a Host Aggregate

1. Set the **baremetal** property to **true** on the baremetal flavor.

```
$ openstack flavor set baremetal --property baremetal=true
```

2. Set the **baremetal** property to **false** on the flavors used for virtual instances.

```
$ openstack flavor set FLAVOR_NAME --property baremetal=false
```

3. Create a host aggregate called **baremetal-hosts**:

```
$ openstack aggregate create --property baremetal=true baremetal-hosts
```

4. Add each controller node to the **baremetal-hosts** aggregate:

```
$ openstack aggregate add host baremetal-hosts HOSTNAME
```

NOTE

If you have created a composable role with the **NovaIronic** service, add all the nodes with this service to the **baremetal-hosts** aggregate. By default, only the controller nodes have the **NovaIronic** service.

5. Create a host aggregate called **virtual-hosts**:

```
$ openstack aggregate create --property baremetal=false virtual-hosts
```

6. Add each compute node to the **virtual-hosts** aggregate:

```
$ openstack aggregate add host virtual-hosts HOSTNAME
```

7. If you did not add the following Compute filter scheduler when deploying the overcloud, add it now to the existing list under **scheduler_default_filters** in */etc/nova/nova.conf*:

```
AggregateInstanceExtraSpecsFilter
```

CHAPTER 5. ADMINISTERING BARE METAL NODES

This chapter describes how to provision a physical machine on an enrolled bare metal node. Instances can be launched either from the command line or from the OpenStack dashboard.

5.1. LAUNCHING AN INSTANCE USING THE COMMAND LINE INTERFACE

Use the **openstack** command line interface to deploy a bare metal instance.

Deploying an Instance on the Command Line

1. Set up the shell to access Identity as the administrative user:

   ```
   $ source ~/overcloudrc
   ```

2. Deploy the instance:

   ```
   $ openstack server create \
     --nic net-id=NETWORK_UUID \
     --flavor baremetal \
     --image IMAGE_UUID \
     INSTANCE_NAME
   ```

   Replace the following values:

   - Replace *NETWORK_UUID* with the unique identifier for the network that was created for use with the Bare Metal service.
   - Replace *IMAGE_UUID* with the unique identifier for the disk image that was uploaded to the Image service.
   - Replace *INSTANCE_NAME* with a name for the bare metal instance.

   To assign the instance to a security group, include **--security-group SECURITY_GROUP**, replacing *SECURITY_GROUP* with the name of the security group. Repeat this option to add the instance to multiple groups. For more information on security group management, see the Users and Identity Management Guide .

3. Check the status of the instance:

   ```
   $ openstack server list --name INSTANCE_NAME
   ```

5.2. LAUNCH AN INSTANCE USING THE DASHBOARD

Use the dashboard graphical user interface to deploy a bare metal instance.

Deploying an Instance in the Dashboard

1. Log in to the dashboard at **http[s]://*DASHBOARD_IP*/dashboard**.

2. Click **Project > Compute > Instances**

3. Click **Launch Instance**.
   - In the **Details** tab, specify the **Instance Name** and select **1** for **Count**.
   - In the **Source** tab, select an **Image** from **Select Boot Source**, then click the **+** (plus) symbol to select an operating system disk image. The chosen image will move to **Allocated**.
   - In the **Flavor** tab, select **baremetal**.
   - In the **Networks** tab, use the **+** (plus) and **-** (minus) buttons to move required networks from **Available** to **Allocated**. Ensure that the shared network created for the Bare Metal service is selected here.
   - If you would like to assign the instance to a security group, in the **Security Groups** tab, use the arrow to move the group to **Allocated**.
4. Click **Launch Instance**.

5.3. CONFIGURE PORT GROUPS IN THE BARE METAL PROVISIONING SERVICE

NOTE

Port group functionality for bare metal nodes is available in this release as a **Technology Preview**, and therefore is not fully supported by Red Hat. It should only be used for testing, and should not be deployed in a production environment. For more information about Technology Preview features, see Scope of Coverage Details.

Port groups (bonds) provide a method to aggregate multiple network interfaces into a single 'bonded' interface. Port group configuration always takes precedence over an individual port configuration.

If a port group has a physical network, then all the ports in that port group should have the same physical network. The Bare Metal Provisioning service supports configuration of port groups in the instances using **configdrive**.

NOTE

Bare Metal Provisioning service API version 1.26 supports port group configuration.

5.3.1. Configure the Switches

To configure port groups in a Bare Metal Provisioning deployment, you need to configure them on the switches manually. You need to make sure that the mode and properties on the switch correspond to the mode and properties on the bare metal side as the naming can vary on the switch.

NOTE

You cannot use port groups for provisioning and cleaning if you need to boot a deployment using iPXE.

Port group fallback allows all the ports in a port group to fallback to individual switch ports when a connection fails. Based on whether a switch supports port group fallback or not, you can use the ``--support-standalone-ports` and ``--unsupport-standalone-ports` options.

5.3.2. Configure Port Groups in the Bare Metal Provisioning Service

1. Create a port group by specifying the node to which it belongs, its name, address, mode, properties and whether or not it supports fallback to standalone ports.

   ```
   # openstack baremetal port group create --node NODE_UUID --name NAME --address
   MAC_ADDRESS --mode MODE  --property miimon=100 --property
   xmit_hash_policy="layer2+3" --support-standalone-ports
   ```

   You can also update a port group using the **openstack baremetal port group set** command.

   If you do not specify an address, the deployed instance’s port group address will be the same as the OpenStack Networking port. The port group will not be configured if the **neutron** port is not attached.

   During interface attachment, port groups have a higher priority than the ports, so they will be used first. Currently, it is **not** possible to specify whether a port group or a port is desired in an interface attachment request. Port groups that do not have any ports will be ignored.

   NOTE

   Port groups have to be configured manually in standalone mode either in the image or by generating the **configdrive** and adding it to the node’s **instance_info**. Make sure you have **cloud-init** version 0.7.7 or later for the port group configuration to work.

2. Associate a port with a port group:

   - During port creation:

     ```
     # openstack baremetal port create --node NODE_UUID --address MAC_ADDRESS --
     port-group test
     ```

   - During port update:

     ```
     # openstack baremetal port set PORT_UUID --port-group PORT_GROUP_UUID
     ```

3. Boot an instance by providing an image that has **cloud-init** or supports bonding.
   To check if the port group has been set up properly, run the following command:

   ```
   # cat /proc/net/bonding/bondX
   ```

   Here, **X** is a number autogenerated by **cloud-init** for each configured port group, starting with a **0** and incremented by one for each configured port group.

5.4. DETERMINING THE HOST TO IP ADDRESS MAPPING

You can use the following commands to determine which IP addresses are assigned to which host and also to which bare metal node.

This feature allows you to know the host to IP mapping from the undercloud without needing to access the hosts directly.

```
(undercloud) [stack@host01 ~]$ openstack stack output show overcloud HostsEntry --max-width 80

+--------------+---------------------------------------------------------------+
| Field        | Value                                                         |
+--------------+---------------------------------------------------------------+
| description  | The content that should be appended to your /etc/hosts if you |
|              | want to get                                                   |
|              | hostname-based access to the deployed nodes (useful for       |
|              | testing without                                               |
|              | setting up a DNS).                                            |
|              |                                                               |
| output_key   | HostsEntry                                                    |
| output_value | 172.17.0.10 overcloud-controller-0.localdomain overcloud-     |
|              | controller-0                                                  |
|              | 10.8.145.18 overcloud-controller-0.external.localdomain       |
|              | overcloud-controller-0.external                               |
|              | 172.17.0.10 overcloud-controller-0.internalapi.localdomain    |
|              | overcloud-controller-0.internalapi                            |
|              | 172.18.0.15 overcloud-controller-0.storage.localdomain        |
|              | overcloud-controller-0.storage                                |
|              | 172.21.2.12 overcloud-controller-0.storagemgmt.localdomain    |
|              | overcloud-controller-0.storagemgmt                            |
|              | 172.16.0.15 overcloud-controller-0.tenant.localdomain         |
|              | overcloud-controller-0.tenant                                 |
|              | 10.8.146.13 overcloud-controller-0.management.localdomain     |
|              | overcloud-controller-0.management                             |
|              | 10.8.146.13 overcloud-controller-0.ctlplane.localdomain       |
|              | overcloud-controller-0.ctlplane                               |
|              |                                                               |
|              | 172.17.0.21 overcloud-compute-0.localdomain overcloud-        |
|              | compute-0                                                     |
|              | 10.8.146.12 overcloud-compute-0.external.localdomain          |
|              | overcloud-compute-0.external                                  |
|              | 172.17.0.21 overcloud-compute-0.internalapi.localdomain       |
|              | overcloud-compute-0.internalapi                               |
|              | 172.18.0.20 overcloud-compute-0.storage.localdomain           |
|              | overcloud-compute-0.storage                                   |
|              | 10.8.146.12 overcloud-compute-0.storagemgmt.localdomain       |
|              | overcloud-compute-0.storagemgmt                               |
|              | 172.16.0.16 overcloud-compute-0.tenant.localdomain overcloud- |
|              | compute-0.tenant                                              |
|              | 10.8.146.12 overcloud-compute-0.management.localdomain        |
|              | overcloud-compute-0.management                                |
|              | 10.8.146.12 overcloud-compute-0.ctlplane.localdomain          |
|              | overcloud-compute-0.ctlplane                                  |
|              |                                                               |
|              |                                                               |
|              |                                                               |
|              |                                                               |
|              | 10.8.145.16  overcloud.localdomain                            |
|              | 10.8.146.7  overcloud.ctlplane.localdomain                    |
|              | 172.17.0.19  overcloud.internalapi.localdomain                |
|              | 172.18.0.19  overcloud.storage.localdomain                    |
|              | 172.21.2.16  overcloud.storagemgmt.localdomain                |
+--------------+---------------------------------------------------------------+
```

To filter a particular host:

```
(undercloud) [stack@host01 ~]$ openstack stack output show overcloud HostsEntry -c output_value
-f value | grep overcloud-controller-0

172.17.0.12 overcloud-controller-0.localdomain overcloud-controller-0
10.8.145.18 overcloud-controller-0.external.localdomain overcloud-controller-0.external
172.17.0.12 overcloud-controller-0.internalapi.localdomain overcloud-controller-0.internalapi
172.18.0.12 overcloud-controller-0.storage.localdomain overcloud-controller-0.storage
172.21.2.13 overcloud-controller-0.storagemgmt.localdomain overcloud-controller-0.storagemgmt
172.16.0.19 overcloud-controller-0.tenant.localdomain overcloud-controller-0.tenant
10.8.146.13 overcloud-controller-0.management.localdomain overcloud-controller-0.management
10.8.146.13 overcloud-controller-0.ctlplane.localdomain overcloud-controller-0.ctlplane
```

To map the hosts to bare metal nodes:

```
(undercloud) [stack@host01 ~]$ openstack baremetal node list --fields uuid name instance_info -f
json
[
  {
    "UUID": "c0d2568e-1825-4d34-96ec-f08bbf0ba7ae",
    "Instance Info": {
      "root_gb": "40",
      "display_name": "overcloud-compute-0",
      "image_source": "24a33990-e65a-4235-9620-9243bcff67a2",
      "capabilities": "{\"boot_option\": \"local\"}",
      "memory_mb": "4096",
      "vcpus": "1",
      "local_gb": "557",
      "configdrive": "******",
      "swap_mb": "0",
      "nova_host_id": "host01.lab.local"
    },
    "Name": "host2"
  },
  {
    "UUID": "8c3faec8-bc05-401c-8956-99c40cdea97d",
    "Instance Info": {
      "root_gb": "40",
      "display_name": "overcloud-controller-0",
      "image_source": "24a33990-e65a-4235-9620-9243bcff67a2",
      "capabilities": "{\"boot_option\": \"local\"}",
      "memory_mb": "4096",
      "vcpus": "1",
      "local_gb": "557",
      "configdrive": "******",
      "swap_mb": "0",
      "nova_host_id": "host01.lab.local"
    },
    "Name": "host3"
  }
]
```

5.5. ATTACHING AND DETACHING A VIRTUAL NETWORK INTERFACE

The Bare Metal Provisioning service has an API to manage the mapping between virtual network interfaces, for example, the ones used in the OpenStack Networking service and the physical interfaces (NICs). These interfaces are configurable for each Bare Metal Provisioning node, allowing you to set the virtual network interface (VIF) to physical network interface (PIF) mapping logic using the **openstack baremetal node vif*** commands.

The following example procedure describes the steps to attach and detach VIFs.

1. List the VIF IDs currently connected to the bare metal node:

```
$ openstack baremetal node vif list baremetal-0
+--------------------------------------+
| ID                                   |
+--------------------------------------+
| 4475bc5a-6f6e-466d-bcb6-6c2dce0fba16 |
+--------------------------------------+
```

2. After the VIF is attached, the Bare Metal service updates the virtual port in the OpenStack Networking service with the actual MAC address of the physical port.
   This can be checked using the following command:

```
$ openstack port show 4475bc5a-6f6e-466d-bcb6-6c2dce0fba16 -c mac_address -c fixed_ips
+-------------+-----------------------------------------------------------------------------+
| Field       | Value                                                                       |
+-------------+-----------------------------------------------------------------------------+
| fixed_ips   | ip_address='192.168.24.9', subnet_id='1d11c677-5946-4733-87c3-
23a9e06077aa' |
| mac_address | 00:2d:28:2f:8d:95                                                           |
+-------------+-----------------------------------------------------------------------------+
```

3. Create a new port on the network where you have created the **baremetal-0** node:

```
$ openstack port create --network baremetal --fixed-ip ip-address=192.168.24.24 baremetal-
0-extra
```

4. Remove a port from the instance:

```
$ openstack server remove port overcloud-baremetal-0 4475bc5a-6f6e-466d-bcb6-
6c2dce0fba16
```

5. Check that the IP address no longer exists on the list:

```
$ openstack server list
```

6. Check if there are VIFs attached to the node:

```
$ openstack baremetal node vif list baremetal-0
$ openstack port list
```

7. Add the newly created port:

```
$ openstack server add port overcloud-baremetal-0 baremetal-0-extra
```

8. Verify the new IP address shows the new port:

```
$ openstack server list
+--------------------------------------+-------------------------+--------+------------------------+----------
------+---------+
| ID                                   | Name                    | Status | Networks               | Image          |
Flavor  |
+--------------------------------------+-------------------------+--------+------------------------+----------
------+---------+
| 53095a64-1646-4dd1-bbf3-b51cbcc38789 | overcloud-controller-2  | ACTIVE |
ctlplane=192.168.24.7  | overcloud-full | control |
| 3a1bc89c-5d0d-44c7-a569-f2a3b4c73d65 | overcloud-controller-0  | ACTIVE |
ctlplane=192.168.24.8  | overcloud-full | control |
| 6b01531a-f55d-40e9-b3a2-6d02be0b915b | overcloud-controller-1  | ACTIVE |
ctlplane=192.168.24.16 | overcloud-full | control |
| c61cc52b-cc48-4903-a971-073c60f53091 | overcloud-novacompute-0overcloud-baremetal-
0 | ACTIVE | ctlplane=192.168.24.24 | overcloud-full | compute |
+--------------------------------------+-------------------------+--------+------------------------+----------
------+---------+
```

9. Check if the VIF ID is the UUID of the new port:

```
$ openstack baremetal node vif list baremetal-0
+--------------------------------------+
| ID                                   |
+--------------------------------------+
| 6181c089-7e33-4f1c-b8fe-2523ff431ffc |
+--------------------------------------+
```

10. Check if the OpenStack Networking port MAC address is updated and matches one of the Bare Metal service ports:

```
$ openstack port show 6181c089-7e33-4f1c-b8fe-2523ff431ffc -c mac_address -c fixed_ips
+-------------+-----------------------------------------------------------------------------+
| Field       | Value                                                                       |
+-------------+-----------------------------------------------------------------------------+
| fixed_ips   | ip_address='192.168.24.24', subnet_id='1d11c677-5946-4733-87c3-
23a9e06077aa' |
| mac_address | 00:2d:28:2f:8d:95                                                           |
+-------------+-----------------------------------------------------------------------------+
```

11. Reboot the bare metal node so it recognizes the new IP address:

```
$ openstack server reboot overcloud-baremetal-0
```

After detaching or attaching interfaces, the bare metal OS should remove, add, or modify the network interfaces that have changed. When you replace a port, a DHCP request obtains the new IP address, but this may take some time since the old DHCP lease is still valid. The simplest way to initiate these changes immediately is to reboot the bare metal host.

5.6. CONFIGURING NOTIFICATIONS FOR THE BARE METAL SERVICE

You can configure the bare metal service to display notifications for different events that occur within the service. These notifications can be used by external services for billing purposes, monitoring a data store, and so on. This section describes how to enable these notifications.

To enable notifications for the baremetal service, you need to set the following options in your **ironic.conf** configuration file.

- The **notification_level** option in the **[DEFAULT]** section determines the minimum priority level for which notifications are sent. The values for this option can be set to **debug**, **info**, **warning**, **error**, or **critical**. If the option is set to **warning**, all notifications with priority level **warning**, **error**, or **critical** are sent, but not notifications with priority level **debug** or **info**. If this option is not set, no notifications will be sent. The priority level of each available notification is documented below.

- The **transport_url** option in the **[oslo_messaging_notifications]** section determines the message bus used when sending notifications. If this is not set, the default transport used for RPC is used.

All notifications are emitted on the **ironic_versioned_notifications** topic in the message bus. Generally, each type of message that traverses the message bus is associated with a topic describing what the message is about.

NOTE

The notifications can be lost and there is no guarantee that a notification will make it across the message bus to the end user.

CHAPTER 6. USE BARE METAL NODES AS INSTANCES

This use case allows you to deploy an instance that uses a bare metal node as the underlying hardware. Sahara performs two internal tasks when creating a big data cluster:

1. Heat is used to create the instances, including the required network.
2. When the instances are ready (**openstack server list** will be **ACTIVE**), sahara connects to each node and applies the configuration for the specified big data plugin. This can include installing additional software, spawning services, and other tasks, until the big data instance is ready.

6.1. PREREQUISITES

- Use the default parameters when deploying Bare Metal Provisioning (ironic) and Data Processing (sahara) on the overcloud.
- All the bare metal nodes must be grouped under a predefined flavor (referred as below as the **baremetal_flavor**).
- Mixed configurations combining virtual and bare metal nodes have not been tested, and may not be supported.

Typically, virtual instances are usually connected to a private project network and then accessible through a floating IP pool on a public network). However, an issue could arise if the bare metal machines managed by ironic are accessible only through a single network. As a result, the sahara cluster should be configured to not use a floating IP address pool, but to only use that network. This issue is not limited to bare metal nodes, and could also arise when sahara is used with virtual machines only.

6.2. GENERATE THE IMAGES

You may need to generate new images for the bare metal nodes using **sahara-image-elements** (including the additional **baremetal** switch). If doing so, you must also generate the kernel and initrd images. However, you may not need to generate the bare metal images at all, as the images typically generated by **sahara-image-elements** do work as full-disk images. The bare metal images may be needed for the **MapR** plugin, because the flavor requires an ephemeral disk and that further requires a partition image. There is currently a known issue for the generation script that prevents the creation of bare metal images. This is expected to be resolved in a future update.

Once the images are generated, you will need to upload them to glance, and then register them in sahara.

6.3. CREATE THE CLUSTER

In this example, the CDH plugin is used to demonstrate a testing scenario:

1. Create a typical set of CDH node group templates and cluster templates. However, for this use case you will need to specify the new **baremetal_flavor**, and you might not require a floating IP address pool. For example, you might allocate:
   - 1x manager
   - 1x master-core
   - 1x master-additional

- 1x worker-nm-dn

2. Enable **dfs_replication** by setting it to **1**.
3. Set the flavor to **baremetal_flavor**.
4. Create a cluster: The resulting cluster should initialize successfully, and the instances you deploy should use the bare metal nodes.

CHAPTER 7. ML2 NETWORKING-ANSIBLE

This section contains information on the **networking-ansible** ML2 driver in OpenStack Networking (neutron), integration with OpenStack Bare Metal (ironic), and instructions on enabling and configuring this driver on an overcloud.

7.1. MODULAR LAYER 2 (ML2) NETWORKING-ANSIBLE

OpenStack Networking (neutron) contains **networking-ansible**, which is an ML2 driver that uses Ansible Engine Networking to manage network switches. This driver also integrates with OpenStack Bare Metal (ironic) to configure VLANs on switch ports for the bare metal guests. This means that the overcloud deployment process triggers the **networking-ansible** driver to configure any bare metal guests that use a VLAN neutron network and their respective the physical switch.

The current **networking-ansible** driver includes the following functionality:

- Define a VLAN on the switch when creating a network in OpenStack
- Assign a VLAN to an access port on the switch when creating or updating a port in OpenStack
- Remove a VLAN from an access port on the switch when deleting a port in OpenStack

7.2. NETWORKING REQUIREMENTS FOR NETWORKING-ANSIBLE

The following list outlines the networking requirements to enable **networking-ansible** functionality.

- A network switch with Ansible Network Automation support. Red Hat currently only supports the use of Juniper Networks (**junos**) switches.
- The network switch also requires an SSH user so that Ansible Network Automation can interact with the device. This user requires certain permissions on the switch:
  - Access mode
  - Assign a VLAN to a port
  - Create VLANs

  For security purposes, do not provide the SSH user with administrator access to the switch.
- If managing VLAN manually, prepare the VLANs you intend the switch to use, which involves creating the VLANs for the respective switch ports. This requirement is based on the **manage_vlans** option when enabling the networking-ansible functionality.
- The network switch ports reserved for bare metal guests initially require configuration to connect to the dedicated network for introspection. Beyond this, these ports require no additional configuration.

7.3. OPENSTACK BARE METAL (IRONIC) REQUIREMENTS FOR NETWORKING-ANSIBLE

The **networking-ansible** driver integrates with the Openstack Bare Metal (ironic) service. To ensure successful integration, deploy the ironic service to your overcloud with the following recommendations:

- The overcloud requires a provisioning network. Use one of the following options:

  - A bridged network for Ironic services.
  - A custom composable network for Ironic services.

  For more examples of configuring the provisioning network, see Chapter 3, *Deploying an Overcloud with the Bare Metal Service*.

- The overcloud requires a tenant network for the bare metal systems to use after the provisioning process. The examples in this guide use the default **baremetal** network mapped to a bridge named **br-baremetal**. This network also requires a range of VLAN IDs. The following Heat parameters set these values to suit examples in this guide:

  ```
  parameter_defaults:
    NeutronNetworkVLANRanges: baremetal:1200:1299
    NeutronFlatNetworks: datacentre,baremetal
    NeutronBridgeMappings: datacentre:br-ex,baremetal:br-baremetal
  ```

For more information on deploying OpenStack Bare Metal (ironic), see Chapter 3, *Deploying an Overcloud with the Bare Metal Service*.

7.4. ENABLING NETWORKING-ANSIBLE ML2 FUNCTIONALITY

This procedure contains information on how to enable the **networking-ansible** ML2 driver in your overcloud. This involves adding two environment files to your deployment:

/usr/share/openstack-tripleo-heat-templates/environments/neutron-ml2-ansible.yaml

This file enables the **networking-ansible** driver and sets the network type to **vlan**. This file already exists in the core Heat template collection.

/home/stack/templates/ml2-ansible-hosts.yaml

This file contains details about your switches. You create this file manually.

Procedure

1. Create the **/home/stack/templates/ml2-ansible-hosts.yaml** and add the following initial content:

   ```
   parameter_defaults:
     ML2HostConfigs:
   ```

2. Include a **dict** definition in the **ML2HostConfigs** parameter for each switch in your environment. Each **dict** key is the name of the switch and defines a specific **ansible:[switchname]** section in your OpenStack Networking (neutron) ML2 configuration. The **dict** value for each switch contains the switch details. For example, to configure three switches, add three switch keys to the **ML2HostConfigs** parameter:

   ```
   parameter_defaults:
     ML2HostConfigs:
       switch1:
         [SWITCH DETAILS]
       switch2:
         [SWITCH DETAILS]
       switch3:
         [SWITCH DETAILS]
   ```

3. Include the following key value pairs for each switch **dict**:

   ansible_network_os

   (Required) The operating system of the switch. Red Hat currently only supports **junos**.

   ansible_host

   (Required) The IP or hostname of the switch.

   ansible_user

   (Required) The user that Ansible uses to access the switch.

   ansible_ssh_pass

   (Required) The SSH password that Ansible uses to access the switch.

   mac

   Chassis MAC ID of the network device. Used to map the link layer discovery protocol (LLDP) MAC address value to the switch name defined in the **ML2HostConfigs** configuration.

   manage_vlans

   A Boolean variable to define whether OpenStack Networking (neutron) controls the creation and deletion of VLANs on the physical device. This functionality causes the switch to create and delete VLANs with IDs respective to their Neutron networks. If you have predefined these VLANs on the switch and do not require Neutron to create or delete VLANs on the switch, set this parameter to **false**. The default value is **true**.

4. The following example shows how to map these values to their respective keys in a full **ML2HostConfigs** parameter:

```
parameter_defaults:
  ML2HostConfigs:
    switch1:
      ansible_network_os: junos
      ansible_host: 10.0.0.1
      ansible_user: ansible
      ansible_ssh_pass: "p@55w0rd!"
      mac: 01:23:45:67:89:AB
      manage_vlans: false
```

5. Save the **/home/stack/templates/ml2-ansible-hosts.yaml** file.

6. When running the overcloud deployment command, include the **/usr/share/openstack-tripleo-heat-templates/environments/neutron-ml2-ansible.yaml** and **/home/stack/templates/ml2-ansible-hosts.yaml** files with the **-e** option:

```
$ openstack overcloud deploy --templates \
  ...
  -e /usr/share/openstack-tripleo-heat-templates/environments/neutron-ml2-ansible.yaml \
  -e /home/stack/templates/ml2-ansible-hosts.yaml \
  ...
```

The director enables the driver as a part of the OpenStack Networking (neutron) API on the **neutron_api** container.

7.5. CONFIGURING NETWORKS FOR NETWORKING-ANSIBLE

After deploying the overcloud with bare metal provisioning and the networking-ansible driver enabled, create the following networks for your bare metal nodes:

Provisioning network

Bare metal systems use this network for their initial creation.

Tenant network

Bare metal systems switch to this network after provisioning and use this network for internal communication.

Procedure

1. Create the provisioning network and subnet. This depends on the type of provisioning network you are using. See Chapter 4, *Configuring for the Bare Metal Service After Deployment* for information on configuring the provisioning network.

2. Create a tenant network and subnet:

```
$ openstack network create --provider-network-type vlan --provider-physical-network baremetal tenant-net
$ openstack subnet create --network tenant-net --subnet-range 192.168.3.0/24 --allocation-pool start=192.168.3.10,end=192.168.3.20 tenant-subnet
```

Ensure that you set the **--provider-network-type** option to **vlan** to ensure **networking-ansible** functionality.

7.6. CONFIGURING PORTS FOR BARE METAL GUESTS

Bare metal guests require port information to connect to the switch. To accomplish this, set the OpenStack Networking (neutron) port configuration. Use this method if your overcloud does not include bare metal introspection functionality.

Procedure

1. Create a port for the bare metal node. Use the following example command as a basis to create the port:

```
$ openstack baremetal port create [NODE NIC MAC] --node [NODE UUID] \
   --local-link-connection port_id=[SWICH PORT ID] \
   --local-link-connection switch_info=[SWITCH NAME] \
   --local-link-connection switch_id=[SWITCH MAC]
```

Replace the following values in brackets with your own environment details:

[NODE NIC MAC]

The MAC address of the NIC connected to the switch.

--node [NODE UUID]

The UUID of the node that uses the new port.

--local-link-connection port_id=[SWITCH PORT ID]

The port ID on the switch connecting to the bare metal node.

--local-link-connection switch_info=[SWITCH NAME]

The name of the switch connecting to the bare metal node. The switch name must match the respective switch name you defined in the **ML2HostConfigs** parameter.

--local-link-connection switch_id=[SWITCH MAC]

The MAC address of the switch. This must match the respective **mac** value from the switch configuration from the **ML2HostConfigs** parameter. This is an alternative option to using **switch_info**.

7.7. TESTING NETWORKING-ANSIBLE ML2 FUNCTIONS

After the **networking-ansible** configuration for the bare metal node is complete, test the functionality to ensure it works. This involves creating a bare metal workload.

Prerequisites

- An overcloud with OpenStack Baremetal (ironic) services.
- An enabled **networking-ansible** ML2 driver.
- The **ML2HostConfigs** parameter contains switch access details.
- A registered bare metal node.
  1. Configuration of the respective bare metal port used for the node connection on the switch.
- A VLAN-based provisioning network defined in OpenStack Networking (neutron) for initial provisioning.
- A VLAN-based tenant network defined in OpenStack Networking (neutron) for internal communication.

Procedure

1. Create the bare metal system:

   ```
   openstack server create --flavor baremetal --image overcloud-full --key default --network
   tenant-net test1
   ```

   The overcloud initially creates the bare metal system on the provisioning network. When the creation completes, the **networking-ansible** driver changes the port configuration on the switch so that the bare metal system uses the tenant network.

CHAPTER 8. TROUBLESHOOTING THE BARE METAL SERVICE

The following sections contain information and steps that may be useful for diagnosing issues in a setup with the Bare Metal service enabled.

8.1. PXE BOOT ERRORS

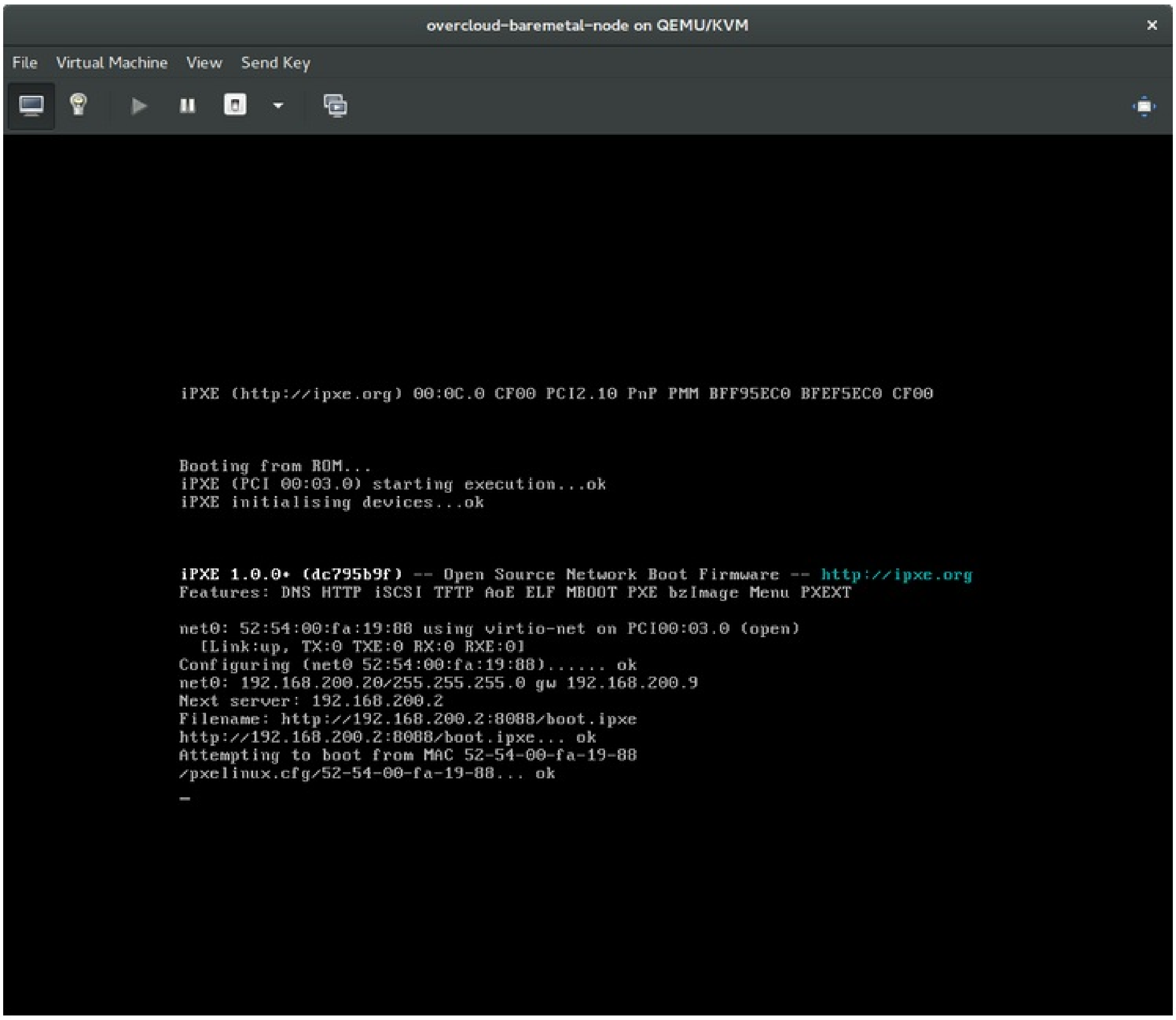
Permission Denied Errors

If you are getting a permission denied error on the console of your Bare Metal service node, make sure you have applied the appropriate SELinux context to the **/httpboot** and **/tftpboot** directories as follows:

```
# semanage fcontext -a -t httpd_sys_content_t "/httpboot(/.*)?"
# restorecon -r -v /httpboot
# semanage fcontext -a -t tftpdir_t "/tftpboot(/.*)?"
# restorecon -r -v /tftpboot
```

Boot Process Freezes at **/pxelinux.cfg/XX-XX-XX-XX-XX-XX**

On the console of your node, if it looks like you are getting an IP address and then the process stops as shown below:



```
iPXE (http://ipxe.org) 00:0C.0 CF00 PCI2.10 PnP PMM BFF95EC0 BFEF5EC0 CF00


Booting from ROM...
iPXE (PCI 00:03.0) starting execution...ok
iPXE initialising devices...ok


iPXE 1.0.0+ (dc795b9f) -- Open Source Network Boot Firmware -- http://ipxe.org
Features: DNS HTTP iSCSI TFTP AoE ELF MBOOT PXE bzImage Menu PXEXT

net0: 52:54:00:fa:19:88 using virtio-net on PCI00:03.0 (open)
  [Link:up, TX:0 TXE:0 RX:0 RXE:0]
Configuring (net0 52:54:00:fa:19:88)...... ok
net0: 192.168.200.20/255.255.255.0 gw 192.168.200.9
Next server: 192.168.200.2
Filename: http://192.168.200.2:8088/boot.ipxe
http://192.168.200.2:8088/boot.ipxe... ok
Attempting to boot from MAC 52-54-00-fa-19-88
/pxelinux.cfg/52-54-00-fa-19-88... ok
_
```

This indicates that you might be using the wrong PXE boot template in your **ironic.conf** file.

```
$ grep ^pxe_config_template ironic.conf
pxe_config_template=$pybasedir/drivers/modules/ipxe_config.template
```

The default template is **pxe_config.template**, so it is easy to miss the *i* to turn this into **ipxe_config.template**.

8.2. LOGIN ERRORS AFTER THE BARE METAL NODE BOOTS

When you try to log in at the login prompt on the console of the node with the **root** password that you set in the configurations steps, but are not able to, it indicates you are not booted in to the deployed image. You are probably stuck in the **deploy-kernel**/**deploy-ramdisk** image and the system has yet to get the correct image.

To fix this issue, verify the PXE Boot Configuration file in the **/httpboot/pxelinux.cfg/MAC_ADDRESS** on the Compute or Bare Metal service node and ensure that all the IP addresses listed in this file correspond to IP addresses on the Bare Metal network.

NOTE

The only network the Bare Metal service node knows about is the Bare Metal network. If one of the endpoints is not on the network, the endpoint will not be able to reach the Bare Metal service node as a part of the boot process.

For example, the kernel line in your file is as follows:

```
kernel http://192.168.200.2:8088/5a6cdbe3-2c90-4a90-b3c6-85b449b30512/deploy_kernel selinux=0
disk=cciss/c0d0,sda,hda,vda iscsi_target_iqn=iqn.2008-10.org.openstack:5a6cdbe3-2c90-4a90-b3c6-
85b449b30512 deployment_id=5a6cdbe3-2c90-4a90-b3c6-85b449b30512
deployment_key=VWDYDVVEFCQJNOSTO9R67HKUXUGP77CK
ironic_api_url=http://192.168.200.2:6385 troubleshoot=0 text nofb nomodeset vga=normal
boot_option=netboot ip=${ip}:${next-server}:${gateway}:${netmask} BOOTIF=${mac}  ipa-api-
url=http://192.168.200.2:6385 ipa-driver-name=pxe_ipmitool boot_mode=bios initrd=deploy_ramdisk
coreos.configdrive=0 || goto deploy
```

Value in the above example **kernel** line	Corresponding information
http://192.168.200.2:8088	Parameter **http_url** in **/etc/ironic/ironic.conf** file. This IP address must be on the Bare Metal network.
5a6cdbe3-2c90-4a90-b3c6-85b449b30512	UUID of the baremetal node in **ironic node-list**.
deploy_kernel	This is the deploy kernel image in the Image service that is copied down as **/httpboot/<NODE_UUID>/deploy_kernel**.

Value in the above example **kernel** line	Corresponding information
http://192.168.200.2:6385	Parameter **api_url** in **/etc/ironic/ironic.conf** file. This IP address must be on the Bare Metal network.
pxe_impitool	The IPMI Driver in use by the Bare Metal service for this node.
deploy_ramdisk	This is the deploy ramdisk image in the Image service that is copied down as **/httpboot/<NODE_UUID>/deploy_ramdisk**.

If a value does not correspond between the **/httpboot/pxelinux.cfg/MAC_ADDRESS** and the **ironic.conf** file:

1. Update the value in the **ironic.conf** file
2. Restart the Bare Metal service
3. Re-deploy the Bare Metal instance

8.3. BOOT-TO-DISK ERRORS ON DEPLOYED NODES

With certain hardware, you might experience a problem with deployed nodes where the nodes cannot boot from disk during successive boot operations as part of a deployment. This usually happens because the BMC does not honor the persistent boot settings that director requests on the nodes. Instead, the nodes boot from a PXE target.

In this case, you must update the boot order in the BIOS of the nodes. Set the HDD to be the first boot device, and then PXE as a later option, so that the nodes boot from disk by default, but can boot from the network during introspection or deployment as necessary.

NOTE

This error mostly applies to nodes that use LegacyBIOS firmware.

8.4. THE BARE METAL SERVICE IS NOT GETTING THE RIGHT HOSTNAME

If the Bare Metal service is not getting the right hostname, it means that **cloud-init** is failing. To fix this, connect the Bare Metal subnet to a router in the OpenStack Networking service. The requests to the meta-data agent should now be routed correctly.

8.5. INVALID OPENSTACK IDENTITY SERVICE CREDENTIALS WHEN EXECUTING BARE METAL SERVICE COMMANDS

If you are having trouble authenticating to the Identity service, check the **identity_uri** parameter in the **ironic.conf** file and make sure you remove the **/v2.0** from the **keystone** AdminURL. For example, **identity_uri** should be set to **http://IP:PORT**.

8.6. HARDWARE ENROLLMENT

Issues with enrolled hardware can be caused by incorrect node registration details. Ensure that property names and values have been entered correctly. Incorrect or mistyped property names will be successfully added to the node's details, but will be ignored.

Update a node's details. This example updates the amount of memory the node is registered to use to 2 GB:

```
$ openstack baremetal node set --property memory_mb=2048 NODE_UUID
```

8.7. *NO VALID HOST* ERRORS

If the Compute scheduler cannot find a suitable Bare Metal node on which to boot an instance, a **NoValidHost** error can be seen in ***/var/log/nova/nova-conductor.log*** or immediately upon launch failure in the dashboard. This is usually caused by a mismatch between the resources Compute expects and the resources the Bare Metal node provides.

1. Check the hypervisor resources that are available:

   ```
   $ openstack hypervisor stats show
   ```

   The resources reported here should match the resources that the Bare Metal nodes provide.

2. Check that Compute recognizes the Bare Metal nodes as hypervisors:

   ```
   $ openstack hypervisor list
   ```

   The nodes, identified by UUID, should appear in the list.

3. Check the details for a Bare Metal node:

   ```
   $ openstack baremetal node list
   $ openstack baremetal node show NODE_UUID
   ```

   Verify that the node's details match those reported by Compute.

4. Check that the selected flavor does not exceed the available resources of the Bare Metal nodes:

   ```
   $ openstack flavor show FLAVOR_NAME
   ```

5. Check the output of **openstack baremetal node list** to ensure that Bare Metal nodes are not in maintenance mode. Remove maintenance mode if necessary:

   ```
   $ openstack baremetal node maintenance unset NODE_UUID
   ```

6. Check the output of **openstack baremetal node list** to ensure that Bare Metal nodes are in an **available** state. Move the node to **available** if necessary:

   ```
   $ openstack baremetal node provide NODE_UUID
   ```

APPENDIX A. BARE METAL DRIVERS

A bare metal node can be configured to use one of the drivers enabled in the Bare Metal service. Each driver is made up of a provisioning method and a power management type. Some drivers require additional configuration. Each driver described in this section uses PXE for provisioning; drivers are listed by their power management type.

You can add drivers with the **IronicEnabledDrivers** parameter in your **ironic.yaml** file. By default, **pxe_ipmitool**, **pxe_drac** and **pxe_ilo** are enabled.

For the full list of supported plug-ins and drivers, see Component, Plug-In, and Driver Support in Red Hat OpenStack Platform.

A.1. INTELLIGENT PLATFORM MANAGEMENT INTERFACE (IPMI)

IPMI is an interface that provides out-of-band remote management features, including power management and server monitoring. To use this power management type, all Bare Metal service nodes require an IPMI that is connected to the shared Bare Metal network. Enable the **pxe_ipmitool** driver, and set the following information in the node's **driver_info**:

- **ipmi_address** - The IP address of the IPMI NIC.
- **ipmi_username** - The IPMI user name.
- **ipmi_password** - The IPMI password.

A.2. DELL REMOTE ACCESS CONTROLLER (DRAC)

DRAC is an interface that provides out-of-band remote management features, including power management and server monitoring. To use this power management type, all Bare Metal service nodes require a DRAC that is connected to the shared Bare Metal network. Enable the **pxe_drac** driver, and set the following information in the node's **driver_info**:

- **drac_address** - The IP address of the DRAC NIC.
- **drac_username** - The DRAC user name.
- **drac_password** - The DRAC password.

A.3. INTEGRATED REMOTE MANAGEMENT CONTROLLER (IRMC)

iRMC from Fujitsu is an interface that provides out-of-band remote management features including power management and server monitoring. To use this power management type on a Bare Metal service node, the node requires an iRMC interface that is connected to the shared Bare Metal network. Enable the **pxe_irmc** driver, and set the following information in the node's **driver_info**:

- **irmc_address** - The IP address of the iRMC interface NIC.
- **irmc_username** - The iRMC user name.
- **irmc_password** - The iRMC password.

To use IPMI to set the boot mode or SCCI to get sensor data, you must complete the following additional steps:

1. Enable the sensor method in ***ironic.conf***:

   ```
   $ openstack-config --set /etc/ironic/ironic.conf \
     irmc sensor_method METHOD
   ```

   Replace *METHOD* with **scci** or **ipmitool**.

2. If you enabled SCCI, install the **python-scciclient** package:

   ```
   # yum install python-scciclient
   ```

3. Restart the Bare Metal conductor service:

   ```
   # systemctl restart openstack-ironic-conductor.service
   ```

NOTE

To use the iRMC driver, iRMC S4 or higher is required.

A.4. INTEGRATED LIGHTS-OUT (ILO)

iLO from Hewlett-Packard is an interface that provides out-of-band remote management features including power management and server monitoring. To use this power management type, all Bare Metal nodes require an iLO interface that is connected to the shared Bare Metal network. Enable the **pxe_ilo** driver, and set the following information in the node's **driver_info**:

- **ilo_address** - The IP address of the iLO interface NIC.
- **ilo_username** - The iLO user name.
- **ilo_password** - The iLO password.

You must also install the **python-proliantutils** package and restart the Bare Metal conductor service:

```
# yum install python-proliantutils
# systemctl restart openstack-ironic-conductor.service
```

A.5. SSH AND VIRSH

NOTE

The **pxe_ssh** driver has been deprecated in favor of VirtualBMC, which uses the **pxe_ipmitool** driver.

The Bare Metal service can access a host that is running libvirt and use virtual machines as nodes. Virsh controls the power management of the nodes.

IMPORTANT

The SSH driver is for testing and evaluation purposes only. It is not recommended for Red Hat OpenStack Platform enterprise environments.

To use this power management type, the Bare Metal service must have SSH access to an account with full access to the libvirt environment on the host where the virtual nodes will be set up. Enable the **pxe_ssh** driver, and set the following information in the node's **driver_info**:

- **ssh_virt_type** - Set this option to **virsh**.
- **ssh_address** - The IP address of the virsh host.
- **ssh_username** - The SSH user name.
- **ssh_key_contents** - The contents of the SSH private key on the Bare Metal conductor node. The matching public key must be copied to the virsh host.

A.6. VIRTUALBMC

VirtualBMC is a utility which can create a virtual baseboard management controller (BMC). A virtual BMC allows you to control virtual machines with the Intelligent Platform Management Interface (IPMI) protocol, just like a physical machine.

IMPORTANT

VirtualBMC is for testing and evaluation purposes only. It is not recommended for Red Hat OpenStack Platform enterprise environments.

1. On the hypervisor hosting the virtual machines, install VirtualBMC:

```
# yum install python-virtualbmc
```

2. Create a BMC for each virtual machine:

```
$ vbmc add DOMAIN --port PORT_NUMBER --username USERNAME --password
PASSWORD
```

3. Start the virtual BMCs:

```
$ vbmc start DOMAIN
```

The virtual BMCs do not start automatically, so if you reboot the host machine, remember to start the virtual BMCs again.

A.6.1. Migrating from pxe_ssh to VirtualBMC

If you have an existing virtual overcloud using the **pxe_ssh** driver, it is possible to migrate to **pxe_impitool** using Virtual BMC.

1. Install **python-virtualbmc** and create BMCs for each virtual machine by following the steps above.

2. Make sure **pxe_impitool** is enabled (it is enabled by default). If it is not enabled, include it in your **ironic.yaml** file and run the **openstack overcloud deploy** command again to apply the changes.

```
parameter_defaults:
```

```
IronicEnabledDrivers:
    - pxe_ipmitool
    - pxe_drac
    - pxe_ilo
```

3. Update the driver and driver properties on each node:

```
$ openstack baremetal node set NODE \
    --driver pxe_ipmitool \
    --driver-info ipmi_address=IP_ADDRESS \
    --driver-info ipmi_port=PORT \
    --driver-info ipmi_username="USERNAME" \
    --driver-info ipmi_password="PASSWORD"
```

- Replace *NODE* with the name or UUID of the node.
- Replace *IP_ADDRESS* with the IP address of the virtual host.
- Replace *PORT* with the Virtual BMC port.

4. Validate that the node has updated correctly:

```
$ openstack baremetal node validate NODE | grep power
```

- Replace *NODE* with the name or UUID of the node.