Red Hat OpenStack Platform 16.2

Bare Metal Provisioning

Install and configure the Bare Metal Provisioning service (ironic)

Last Updated: 2023-11-09

Red Hat OpenStack Platform 16.2 Bare Metal Provisioning

Install and configure the Bare Metal Provisioning service (ironic)

OpenStack Team
rhos-docs@redhat.com

Legal Notice

Abstract

Install and configure the Bare Metal Provisioning service in the overcloud of a Red Hat OpenStack Platform environment to provision and manage physical machines for cloud users.

Table of Contents

MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see our CTO Chris Wright's message .

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your input on our documentation. Tell us how we can make it better.

Providing documentation feedback in Jira

Use the Create Issue form to provide feedback on the documentation. The Jira issue will be created in the Red Hat OpenStack Platform Jira project, where you can track the progress of your feedback.

1. Ensure that you are logged in to Jira. If you do not have a Jira account, create an account to submit feedback.
2. Click the following link to open a the **Create Issue** page: Create Issue
3. Complete the **Summary** and **Description** fields. In the **Description** field, include the documentation URL, chapter or section number, and a detailed description of the issue. Do not modify any other fields in the form.
4. Click **Create**.

CHAPTER 1. BARE METAL PROVISIONING SERVICE (IRONIC) FUNCTIONALITY

You use the Bare Metal Provisioning service (ironic) components to provision and manage physical machines as bare metal instances for your cloud users. To provision and manage bare metal instances, the Bare Metal Provisioning service interacts with the following Red Hat OpenStack Platform (RHOSP) services in the overcloud:

- The Compute service (nova) provides scheduling, tenant quotas, and a user-facing API for virtual machine instance management. The Bare Metal Provisioning service provides the administrative API for hardware management.
- The Identity service (keystone) provides request authentication and assists the Bare Metal Provisioning service to locate other RHOSP services.
- The Image service (glance) manages disk and instance images and image metadata.
- The Networking service (neutron) provides DHCP and network configuration, and provisions the virtual or physical networks that instances connect to on boot.
- The Object Storage service (swift) exposes temporary image URLs for some drivers.

Bare Metal Provisioning service components

The Bare Metal Provisioning service consists of services, named **ironic-***. The following services are the core Bare Metal Provisioning services:

Bare Metal Provisioning API (ironic-api)

This service provides the external REST API to users. The API sends application requests to the Bare Metal Provisioning conductor over remote procedure call (RPC).

Bare Metal Provisioning conductor (ironic-conductor)

This service uses drivers to perform the following bare metal node management tasks:

- Adds, edits, and deletes bare metal nodes.
- Powers bare metal nodes on and off with IPMI, Redfish, or other vendor-specific protocol.
- Provisions, deploys, and cleans bare metal nodes.

Bare Metal Provisioning inspector (ironic-inspector)

This service discovers the hardware properties of a bare metal node that are required for scheduling bare metal instances, and creates the Bare Metal Provisioning service ports for the discovered ethernet MACs.

Bare Metal Provisioning database

This database tracks hardware information and state.

Message queue

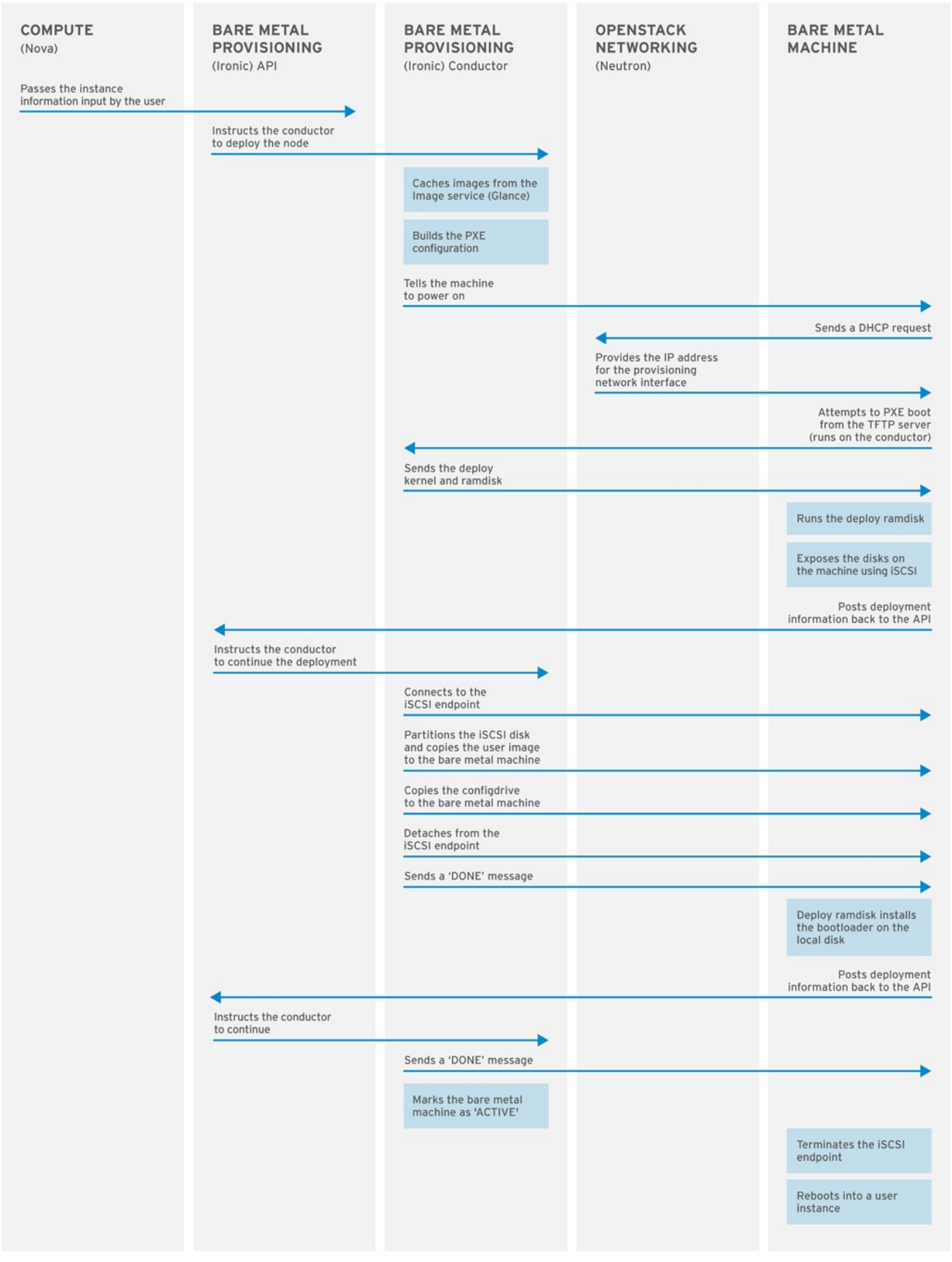
All services use this messaging service to communicate with each other, including implementing the RPC between **ironic-api** and **ironic-conductor**.

Bare Metal Provisioning agent (ironic-python-agent)

This service runs in a temporary ramdisk to provide **ironic-conductor** and **ironic-inspector** services with remote access, in-band hardware control, and hardware introspection.

Provisioning a bare metal instance

The Bare Metal Provisioning service uses iPXE to provision physical machines as bare metal instances. The following diagram outlines how the RHOSP services interact during the provisioning process when a cloud user launches a new bare metal instance with the default drivers.



CHAPTER 2. REQUIREMENTS FOR BARE METAL PROVISIONING

To provide an overcloud where cloud users can launch bare metal instances, your Red Hat OpenStack Platform (RHOSP) environment must have the required hardware and network configuration.

2.1. HARDWARE REQUIREMENTS

The hardware requirements for the bare metal machines that you want to make available to your cloud users for provisioning depend on the operating system. For information about the hardware requirements for Red Hat Enterprise Linux installations, see Product Documentation for Red Hat Enterprise Linux.

All bare metal machines that you want to make available to your cloud users for provisioning must have the following capabilities:

- A NIC to connect to the bare metal network.
- A power management interface, for example, Redfish or IPMI, that is connected to a network that is reachable from the **ironic-conductor** service. By default, **ironic-conductor** runs on all of the Controller nodes, unless you use composable roles and run **ironic-conductor** elsewhere.
- PXE boot on the bare metal network. Disable PXE boot on all other NICs in the deployment.

2.2. NETWORKING REQUIREMENTS

The bare metal network must be a private network for the Bare Metal Provisioning service to use for the following operations:

- The provisioning and management of bare metal machines on the overcloud.
- Cleaning bare metal nodes when a node is unprovisioned.
- Tenant access to the bare metal machines.

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 Provisioning service can serve PXE boot and DHCP requests.

The Bare Metal Provisioning service in the overcloud is designed for a trusted tenant environment because the bare metal machines have direct access to the control plane network of your Red Hat OpenStack Platform (RHOSP) environment. Therefore, the default bare metal network uses a flat network for **ironic-conductor** services.

The default flat provisioning network can introduce security concerns in a customer environment because a tenant can interfere with the control plane network. To prevent this risk, you can configure a custom composable bare metal provisioning network for the Bare Metal Provisioning service that does not have access to the control plane.

The bare metal network must be untagged for provisioning, and must also have access to the Bare Metal Provisioning API. The control plane network, also known as the director provisioning network, is always untagged. Other networks can be tagged.

The Controller nodes that host the Bare Metal Provisioning service must have access to the bare metal network.

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

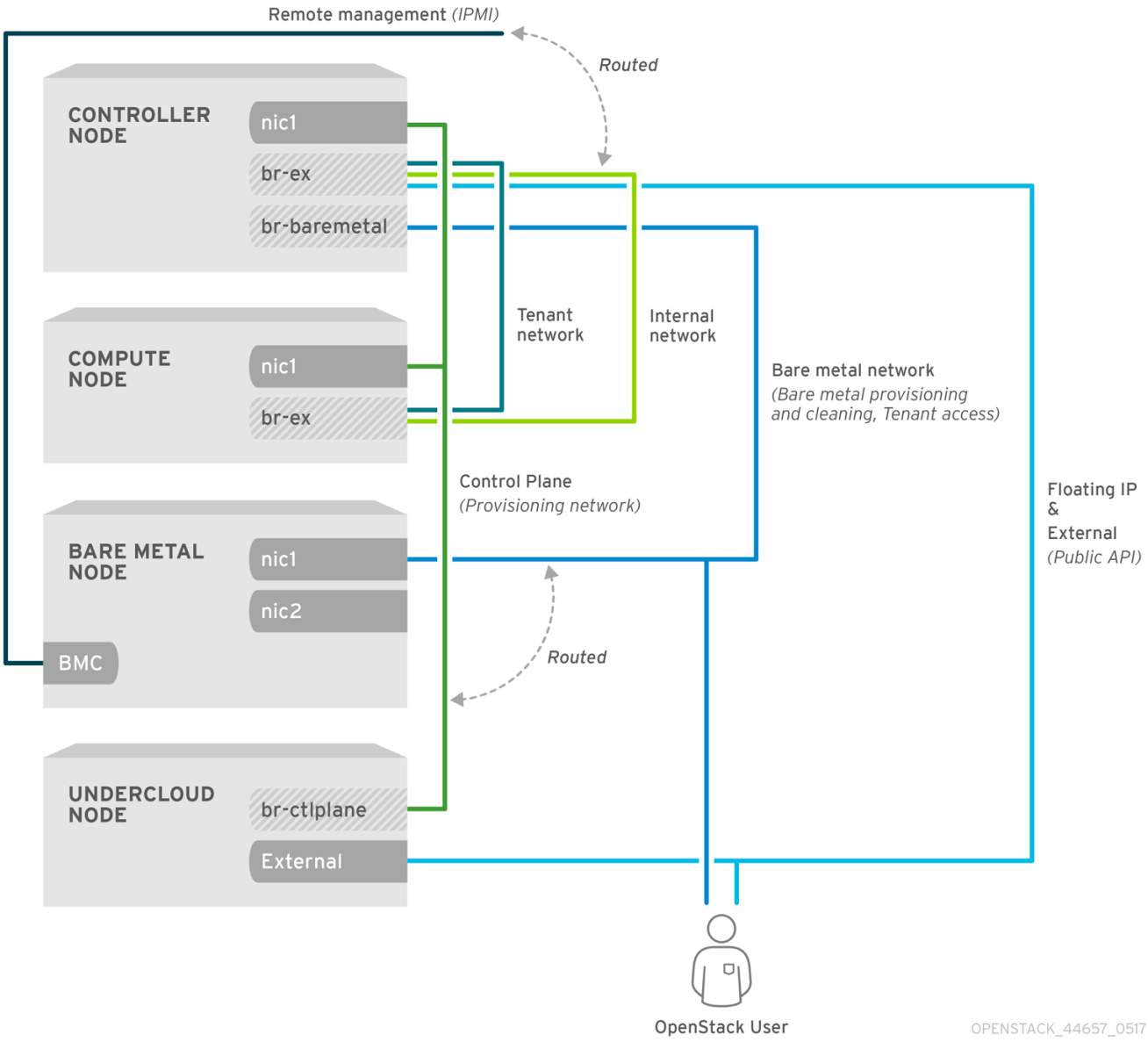
The bare metal network is created by the OpenStack operator. Cloud users have direct access to the public OpenStack APIs, and to the bare metal network. With the default flat bare metal network, cloud users also have indirect access to the control plane.

The Bare Metal Provisioning service uses the bare metal network for node cleaning.

2.2.1. The default bare metal network

In the default Bare Metal Provisioning service deployment architecture, the bare metal network is separate from the control plane network. The bare metal network is a flat network that also acts as the tenant network. This network must route to the Bare Metal Provisioning services on the control plane, known as the director provisioning network. If you define an isolated bare metal network, the bare metal nodes cannot PXE boot.

Default bare metal network architecture diagram



2.2.2. The custom composable bare metal network

When you use a custom composable bare metal network in your Bare Metal Provisioning service

deployment architecture, the bare metal network is a custom composable network that does not have access to the control plane. Use a custom composable bare metal network if you want to limit access to the control plane.

CHAPTER 3. DEPLOYING AN OVERCLOUD WITH THE BARE METAL PROVISIONING SERVICE

To deploy an overcloud with the Bare Metal Provisioning service (ironic), you must create and configure the bare metal network, and configure the overcloud to enable bare metal provisioning.

1. Create the bare metal network. You can reuse the provisioning network interface on the Controller nodes to create a flat network, or you can create a custom network:
   - Configuring the default flat network
   - Configuring a custom IPv4 provisioning network
   - Configuring a custom IPv6 provisioning network
2. Configure the overcloud to enable bare metal provisioning:
   - Configuring the overcloud to enable bare metal provisioning

NOTE

If you use Open Virtual Network (OVN), the Bare Metal Provisioning service is supported only with the DHCP agent defined in the **ironic-overcloud.yaml** file, **neutron-dhcp-agent**. The built-in DHCP server on OVN cannot provision bare metal nodes or serve DHCP for the provisioning networks. To enable iPXE chain loading you must set the **--dhcp-match** tag in dnsmasq, which is not supported by the OVN DHCP server.

Prerequisites

- Your environment meets the minimum requirements. For more information, see Requirements for bare metal provisioning.

3.1. CONFIGURING THE DEFAULT FLAT NETWORK

To use the default flat bare metal network, you reuse the provisioning network interface on the Controller nodes to create a bridge for the Bare Metal Provisioning service (ironic).

Procedure

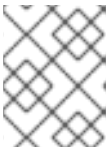
1. Log in to the undercloud as the **stack** user.
2. Source the **stackrc** file:

```
[stack@director ~]$ source ~/stackrc
```

3. Modify the **/home/stack/templates/nic-configs/controller.yaml** file to reuse the provisioning network interface on the Controller nodes, **eth1**, to create a bridge for the bare metal network:

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

```
    name: eth1
  addresses:
    - ip_netmask:
        list_join:
        - /
        - - get_param: ControlPlaneIp
          - get_param: ControlPlaneSubnetCidr
```



NOTE

You cannot VLAN tag the bare metal network when you create it by reusing the provisioning network.

4. Add **br-baremetal** to the **NeutronBridgeMappings** parameter in your **network-environment.yaml** file:

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

5. Add **baremetal** to the list of networks specified by the **NeutronFlatNetworks** parameter in your **network-environment.yaml** file:

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

Next steps

- Configuring the overcloud to enable bare metal provisioning

3.2. CONFIGURING A CUSTOM IPV4 PROVISIONING NETWORK

Create a custom IPv4 provisioning network to provision and deploy the overcloud over IPv4.

Procedure

1. Log in to the undercloud as the **stack** user.
2. Source the **stackrc** file:

```
[stack@director ~]$ source ~/stackrc
```

3. Copy the **network_data.yaml** file to your environment file directory:

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

4. Add a new network for overcloud provisioning to your **network_data.yaml** file:

```
# custom network for overcloud provisioning
- name: OcProvisioning
  name_lower: oc_provisioning
```

```
vip: true
vlan: 205
ip_subnet: '<ipv4_subnet_address>/<ipv4_mask>'
allocation_pools: [{'start': '<ipv4_start_address>', 'end': '<ipv4_end_address>'}]
```

- Replace **<ipv4_subnet_address>** with the IPv4 address of your IPv4 subnet.
- Replace **<ipv4_mask>** with the IPv4 network mask for your IPv4 subnet.
- Replace **<ipv4_start_address>** and **<ipv4_end_address>** with the IPv4 range that you want to use for address allocation.

5. Configure **IronicApiNetwork** and **IronicNetwork** in your **ServiceNetMap** configuration to use the new IPv4 provisioning network:

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

6. Add the new network as an interface to your local Controller NIC configuration file:

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

7. Copy the **roles_data.yaml** file to your environment file directory:

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

8. Add the new network for the controller to your **roles_data.yaml** file:

```
networks:
  ...
  OcProvisioning:
    subnet: oc_provisioning_subnet
```

9. Include the **IronicInspector** service in the **Ironic** role in your **roles_data.yaml** file, if not already present:

```
ServicesDefault:
  OS::TripleO::Services::IronicInspector
```

Next steps

- Configuring the overcloud to enable bare metal provisioning

3.3. CONFIGURING A CUSTOM IPV6 PROVISIONING NETWORK

Create a custom IPv6 provisioning network to provision and deploy the overcloud over IPv6.

Procedure

1. Log in to the undercloud as the **stack** user.
2. Source the **stackrc** file:

```
[stack@director ~]$ source ~/stackrc
```

3. Copy the **network_data.yaml** file to your environment file directory:

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

4. Add a new IPv6 network for overcloud provisioning to your **network_data.yaml** file:

```
# custom network for IPv6 overcloud provisioning
- name: OcProvisioningIPv6
  vip: true
  name_lower: oc_provisioning_ipv6
  vlan: 10
  ipv6: true
  ipv6_subnet: '<ipv6_subnet_address>/<ipv6_prefix>'
  ipv6_allocation_pools: [{'start': '<ipv6_start_address>', 'end': '<ipv6_end_address>'}]
  gateway_ipv6: '<ipv6_gw_address>'
```

   - Replace **<ipv6_subnet_address>** with the IPv6 address of your IPv6 subnet.
   - Replace **<ipv6_prefix>** with the IPv6 network prefix for your IPv6 subnet.
   - Replace **<ipv6_start_address>** and **<ipv6_end_address>** with the IPv6 range that you want to use for address allocation.
   - Replace **<ipv6_gw_address>** with the IPv6 address of your gateway.

5. Create a new file **network_environment_overrides.yaml** in your environment file directory:

```
$ touch /home/stack/templates/network_environment_overrides.yaml
```

6. Configure **IronicApiNetwork** and **IronicNetwork** in your **network_environment_overrides.yaml** file to use the new IPv6 provisioning network:

```
ServiceNetMap:
  IronicApiNetwork: oc_provisioning_ipv6
  IronicNetwork: oc_provisioning_ipv6
```

7. Set the **IronicIpVersion** parameter to **6**:

```
parameter_defaults:
  IronicIpVersion: 6
```

8. Enable the **RabbitIPv6**, **MysqlIPv6**, and **RedisIPv6** parameters:

```
parameter_defaults:
  RabbitIPv6: True
  MysqlIPv6: True
  RedisIPv6: True
```

9. Add the new network as an interface to your local Controller NIC configuration file:

```
network_config:
- type: vlan
  vlan_id:
    get_param: OcProvisioningIPv6NetworkVlanID
  addresses:
  - ip_netmask:
      get_param: OcProvisioningIPv6IpSubnet
```

10. Copy the **roles_data.yaml** file to your environment file directory:

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

11. Add the new network for the Controller role to your **roles_data.yaml** file:

```
networks:
  ...
  - OcProvisioningIPv6
```

12. Include the **IronicInspector** service in the **Ironic** role in your **roles_data.yaml** file, if not already present:

```
ServicesDefault:
  OS::TripleO::Services::IronicInspector
```

Next steps

- Configuring the overcloud to enable bare metal provisioning

3.4. CONFIGURING THE OVERCLOUD TO ENABLE BARE METAL PROVISIONING

Use one of the default templates located in the **/usr/share/openstack-tripleo-heat-templates/environments/services** directory to deploy the overcloud with the Bare Metal Provisioning service (ironic) enabled:

- For deployments that use OVS: **ironic.yaml**
- For deployments that use OVN: **ironic-overcloud.yaml**

You can create a local environment file to override the default configuration, as required by your deployment.

Procedure

1. Create an environment file in your local directory to configure the Bare Metal Provisioning service for your deployment, for example, **ironic-overrides.yaml**.

2. Optional: Configure the type of cleaning that is performed on the bare metal machines before and between provisioning:

```
parameter_defaults:
  IronicCleaningDiskErase: <cleaning_type>
```

   Replace **<cleaning_type>** with one of the following values:

   - **full**: (Default) Performs a full clean.
   - **metadata**: Clean only the partition table. This type of cleaning substantially speeds up the cleaning process. However, because the deployment is less secure in a multi-tenant environment, use this option only in a trusted tenant environment.

3. Optional: Add additional drivers to the default drivers:

```
parameter_defaults:
  IronicEnabledHardwareTypes: ipmi,idrac,ilo,[additional_driver_1],...,[additional_driver_n]
```

   Replace **[additional_driver_1]**, and optionally all drivers up to **[additional_driver_n]**, with the additional drivers you want to enable.

4. To enable bare metal introspection, add the following configuration to your local Bare Metal Provisioning service environment file, **ironic-overrides.yaml**:

```
parameter_defaults:
  IronicInspectorSubnets:
    - ip_range: <ip_range>
  IPAImageURLs: '["http://<ip_address>:<port>/agent.kernel", "http://<ip_address>:
<port>/agent.ramdisk"]'
  IronicInspectorInterface: '<baremetal_interface>'
```

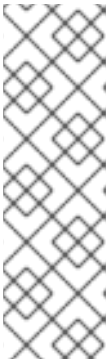
   - Replace **<ip_range>** with the IP ranges for your environments, for example, **192.168.0.100,192.168.0.120**.
   - Replace **<ip_address>:<port>** with the IP address and port of the web server that hosts the IPA kernel and ramdisk. To use the same images that you use on the undercloud, set the IP address to the undercloud IP address, and the port to **8088**. If you omit this parameter, you must include alternatives on each Controller node.
   - Replace **<baremetal_interface>** with the bare metal network interface, for example, **br-baremetal**.

5. Add your new role and custom environment files to the stack with your other environment files and deploy the overcloud:

```
(undercloud)$ openstack overcloud deploy --templates \
  -e [your environment files] \
  -e /home/stack/templates/node-info.yaml \
  -r /home/stack/templates/roles_data.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/network-environment.yaml \
  -e /usr/share/openstack-tripleo-heat-
templates/environments/services/<default_ironic_template> \
```

```
  -e /usr/share/openstack-tripleo-heat-templates/environments/services/ironic-inspector.yaml
\
  -e /home/stack/templates/network_environment_overrides.yaml \
  -n /home/stack/templates/network_data.yaml \
  -e /home/stack/templates/ironic-overrides.yaml
```

- Replace **<default_ironic_template>** with either **ironic.yaml** or **ironic-overcloud.yaml**, depending on the Networking service mechanism driver for your deployment.



NOTE

The order that you pass your environment files to the **openstack overcloud deploy** command is important, as the configuration in the later files takes precedence. Therefore, your environment file that enables and configures bare metal provisioning on your overcloud must be passed to the command after any network configuration files.

For more information on using the **openstack overcloud deploy** command, see Including environment files in an overcloud deployment .

3.5. TESTING THE BARE METAL PROVISIONING 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 verification methods for the Bare Metal Provisioning service:

1. Configure 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 changed the default ironic drivers, ensure that the required drivers are enabled:

```
$ openstack baremetal driver list
```

4. Ensure that the ironic endpoints are listed:

```
$ openstack catalog list
```

3.6. ADDITIONAL RESOURCES

- Deployment command options in the *Director Installation and Usage* guide
- Including environment files in an overcloud deployment in the *Director Installation and Usage* guide
- IPv6 Networking for the Overcloud
- Bare Metal (ironic) Parameters in the *Overcloud Parameters* guide

CHAPTER 4. CONFIGURING THE BARE METAL PROVISIONING SERVICE AFTER DEPLOYMENT

When you have deployed your overcloud with the Bare Metal Provisioning service (ironic), you must prepare your overcloud for bare-metal workloads. To prepare your overcloud for bare-metal workloads and enable your cloud users to create bare-metal instances, complete the following tasks:

- Configure the Networking service (neutron) to integrate with the Bare Metal Provisioning service.
- Configure node cleaning.
- Create the bare-metal flavor and resource class.
- Optional: Create the bare-metal images.
- Add physical machines as bare-metal nodes.
- Optional: Configure Redfish virtual media boot.
- Optional: Create host aggregates to separate physical and virtual machine provisioning.

4.1. CONFIGURING THE NETWORKING SERVICE FOR BARE METAL PROVISIONING

You can configure the Networking service (neutron) to integrate with the Bare Metal Provisioning service (ironic). You can configure the bare-metal network by using one of the following methods:

- Create a single flat bare-metal network for the Bare Metal Provisioning conductor services, **ironic-conductor**. This network must route to the Bare Metal Provisioning services on the control plane network.
- Create a custom composable network to implement Bare Metal Provisioning services in the overcloud.

4.1.1. Configuring the Networking service to integrate with the Bare Metal Provisioning service on a flat network

You can configure the Networking service (neutron) to integrate with the Bare Metal Provisioning service (ironic) by creating a single flat bare-metal network for the Bare Metal Provisioning conductor services, **ironic-conductor**. This network must route to the Bare Metal Provisioning services on the control plane network.

Procedure

1. Log in to the node that hosts the Networking service (neutron) as the **root** user.
2. Source your overcloud credentials file:

```
# source ~/<credentials_file>
```

   - Replace **<credentials_file>** with the name of your credentials file, for example, **overcloudrc**.

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

```
# openstack network create \
  --provider-network-type flat \
  --provider-physical-network <provider_physical_network> \
  --share <network_name>
```

   - Replace **<provider_physical_network>** with the name of the physical network over which you implement the virtual network, which is configured with the parameter **NeutronBridgeMappings** in your **network-environment.yaml** file.
   - Replace **<network_name>** with a name for this network.

4. 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 **<network_name>** with the name of the provisioning network that you created in the previous step.
   - Replace **<network_cidr>** with the Classless Inter-Domain Routing (CIDR) representation of the block of IP addresses that the subnet represents. The block of IP addresses that you specify in the range starting with **<start_ip>** and ending with **<end_ip>** must be 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 acts 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 that starts with **<start_ip>** and ends with **<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 are 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 are allocated.
   - Replace **<subnet_name>** with a name for the subnet.

5. Create a router for the network and subnet to ensure that the Networking service serves metadata requests:

```
# openstack router create <router_name>
```

   - Replace **<router_name>** with a name for the router.

6. Attach the subnet to the new router to enable the metadata requests from **cloud-init** to be served and the node to be configured: :

```
# openstack router add subnet <router_name> <subnet>
```

- Replace **<router_name>** with the name of your router.
- Replace **<subnet>** with the ID or name of the bare-metal subnet that you created in the step 4.

4.1.2. Configuring the Networking service to integrate with the Bare Metal Provisioning service on a custom composable network

You can configure the Networking service (neutron) to integrate with the Bare Metal Provisioning service (ironic) by creating a custom composable network to implement Bare Metal Provisioning services in the overcloud.

Procedure

1. Log in to the undercloud host.
2. Source your overcloud credentials file:

```
$ source ~/<credentials_file>
```

   - Replace **<credentials_file>** with the name of your credentials file, for example, **overcloudrc**.

3. Retrieve the UUID for the provider network that hosts the Bare Metal Provisioning service:

```
(overcloud)$ openstack network show <network_name> -f value -c id
```

   - Replace **<network_name>** with the name of the provider network that you want to use for the bare-metal instance provisioning network.

4. Open your local environment file that configures the Bare Metal Provisioning service for your deployment, for example, **ironic-overrides.yaml**.
5. Configure the network to use as the bare-metal instance provisioning network:

```
parameter_defaults:
  IronicProvisioningNetwork: <network_uuid>
```

   - Replace **<network_uuid>** with the UUID of the provider network retrieved in step 3.

6. Source the **stackrc** undercloud credentials file:

```
$ source ~/stackrc
```

7. To apply the bare-metal instance provisioning network configuration, add your Bare Metal Provisioning environment files to the stack with your other environment files and deploy the overcloud:

```
(undercloud)$ openstack overcloud deploy --templates \
  -e [your environment files] \
  -e /home/stack/templates/node-info.yaml \
  -r /home/stack/templates/roles_data.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/network-environment.yaml \
  -e /usr/share/openstack-tripleo-heat-
```

```
templates/environments/services/<default_ironic_template> \
  -e /usr/share/openstack-tripleo-heat-templates/environments/services/ironic-inspector.yaml
\
  -e /home/stack/templates/network_environment_overrides.yaml \
  -n /home/stack/templates/network_data.yaml \
  -e /home/stack/templates/ironic-overrides.yaml
```

- Replace **<default_ironic_template>** with either **ironic.yaml** or **ironic-overcloud.yaml**, depending on the Networking service mechanism driver for your deployment.

4.2. CLEANING BARE-METAL NODES

The Bare Metal Provisioning service cleans nodes to prepare them for provisioning. You can clean bare-metal nodes by using one of the following methods:

- Automatic: You can configure your overcloud to automatically perform node cleaning when you unprovision a node.
- Manual: You can manually clean individual nodes when required.

4.2.1. Configuring automatic node cleaning

Automatic bare-metal node cleaning runs after you enroll a node, and before the node reaches the **available** provisioning state. Automatic cleaning is run each time the node is unprovisioned.

By default, the Bare Metal Provisioning service uses a network named **provisioning** for node cleaning. However, network names are not unique in the Networking service (neutron), so it is possible for a project to create a network with the same name, which causes a conflict with the Bare Metal Provisioning service. To avoid the conflict, use the network UUID to configure the node cleaning network.

Procedure

1. Log in to the undercloud host.
2. Source your overcloud credentials file:

```
$ source ~/<credentials_file>
```

   - Replace **<credentials_file>** with the name of your credentials file, for example, **overcloudrc**.

3. Retrieve the UUID for the provider network that hosts the Bare Metal Provisioning service:

```
(overcloud)$ openstack network show <network_name> -f value -c id
```

   - Replace **<network_name>** with the name of the network that you want to use for the bare-metal node cleaning network.

4. Open your local environment file that configures the Bare Metal Provisioning service for your deployment, for example, **ironic-overrides.yaml**.
5. Configure the network to use as the node cleaning network:

```
parameter_defaults:
  IronicCleaningNetwork: <network_uuid>
```

- Replace **<network_uuid>** with the UUID of the provider network that you retrieved in step 3.

6. Source the **stackrc** undercloud credentials file:

```
$ source ~/stackrc
```

7. To apply the node cleaning network configuration, add your Bare Metal Provisioning environment files to the stack with your other environment files and deploy the overcloud:

```
(undercloud)$ openstack overcloud deploy --templates \
  -e [your environment files] \
  -e /home/stack/templates/node-info.yaml \
  -r /home/stack/templates/roles_data.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/network-environment.yaml \
  -e /usr/share/openstack-tripleo-heat-
templates/environments/services/<default_ironic_template> \
  -e /usr/share/openstack-tripleo-heat-templates/environments/services/ironic-inspector.yaml
\
  -e /home/stack/templates/network_environment_overrides.yaml \
  -n /home/stack/templates/network_data.yaml \
  -e /home/stack/templates/ironic-overrides.yaml
```

- Replace **<default_ironic_template>** with either **ironic.yaml** or **ironic-overcloud.yaml**, depending on the Networking service mechanism driver for your deployment.

4.2.2. Cleaning nodes manually

You can clean specific nodes manually as required. Node cleaning has two modes:

- Metadata only clean: Removes partitions from all disks on the node. The metadata only mode of cleaning is faster than a full clean, but less secure because it erases only partition tables. Use this mode only on trusted tenant environments.
- Full clean: Removes all data from all disks, using either ATA secure erase or by shredding. A full clean can take several hours to complete.

Procedure

1. Source your overcloud credentials file:

```
$ source ~/<credentials_file>
```

- Replace **<credentials_file>** with the name of your credentials file, for example, **overcloudrc**.

2. Check the current state of the node:

```
$ openstack baremetal node show \
 -f value -c provision_state <node>
```

- Replace **<node>** with the name or UUID of the node to clean.

3. If the node is not in the **manageable** state, then set it to **manageable**:

```
$ openstack baremetal node manage <node>
```

4. Clean the node:

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

- Replace **<node>** with the name or UUID of the node to clean.
- Replace **<clean_mode>** with the type of cleaning to perform on the node:
  - **erase_devices**: Performs a full clean.
  - **erase_devices_metadata**: Performs a metadata only clean.

5. Wait for the clean to complete, then check the status of the node:

- **manageable**: The clean was successful, and the node is ready to provision.
- **clean failed**: The clean was unsuccessful. Inspect the **last_error** field for the cause of failure.

4.3. CREATING FLAVORS FOR LAUNCHING BARE-METAL INSTANCES

You must create flavors that your cloud users can use to request bare-metal instances. You can specify which bare-metal nodes should be used for bare-metal instances launched with a particular flavor by using a resource class. You can tag bare-metal nodes with resource classes that identify the hardware resources on the node, for example, GPUs. The cloud user can select a flavor with the GPU resource class to create an instance for a vGPU workload. The Compute scheduler uses the resource class to identify suitable host bare-metal nodes for instances.

Procedure

1. Source the overcloud credentials file:

```
$ source ~/overcloudrc
```

2. Create a flavor for bare-metal instances:

```
(overcloud)$ openstack flavor create --id auto \
 --ram <ram_size_mb> --disk <disk_size_gb> \
 --vcpus <no_vcpus> baremetal
```

- Replace **<ram_size_mb>** with the RAM of the bare metal node, in MB.
- Replace **<disk_size_gb>** with the size of the disk on the bare metal node, in GB.
- Replace **<no_vcpus>** with the number of CPUs on the bare metal node.

NOTE

These properties are not used for scheduling instances. However, the Compute scheduler does use the disk size to determine the root partition size.

3. Retrieve a list of your nodes to identify their UUIDs:

```
(overcloud)$ openstack baremetal node list
```

4. Tag each bare-metal node with a custom bare-metal resource class:

```
(overcloud)$ openstack baremetal node set \
 --resource-class baremetal.<CUSTOM> <node>
```

- Replace **<CUSTOM>** with a string that identifies the purpose of the resource class. For example, set to **GPU** to create a custom GPU resource class that you can use to tag bare metal nodes that you want to designate for GPU workloads.
- Replace **<node>** with the ID of the bare metal node.

5. Associate the flavor for bare-metal instances with the custom resource class:

```
(overcloud)$ openstack flavor set \
 --property resources:CUSTOM_BAREMETAL_<CUSTOM>=1 \
 baremetal
```

To determine the name of a custom resource class that corresponds to a resource class of a bare-metal node, convert the resource class to uppercase, replace each punctuation mark with an underscore, and prefix with **CUSTOM_**.

NOTE

A flavor can request only one instance of a bare-metal resource class.

6. Set the following flavor properties to prevent the Compute scheduler from using the bare-metal flavor properties to schedule instances:

```
(overcloud)$ openstack flavor set \
 --property resources:VCPU=0 \
 --property resources:MEMORY_MB=0 \
 --property resources:DISK_GB=0 baremetal
```

7. Verify that the new flavor has the correct values:

```
(overcloud)$ openstack flavor list
```

4.4. CREATING IMAGES FOR LAUNCHING BARE-METAL INSTANCES

An overcloud that includes the Bare Metal Provisioning service (ironic) requires two sets of images:

- Deploy images: The deploy images are the **agent.ramdisk** and **agent.kernel** images that the Bare Metal Provisioning agent (**ironic-python-agent**) requires to boot the RAM disk over the

network and copy the user image for the overcloud nodes to the disk. You install the deploy images as part of the undercloud installation. For more information, see Obtaining images for overcloud nodes.

- User images: The images the cloud user uses to provision their bare-metal instances. The user image consists of a **kernel** image, a **ramdisk** image, and a **main** image. The main image is either a root partition, or a whole-disk image:
  - Whole-disk image: An image that contains the partition table and boot loader.
  - Root partition image: Contains only the root partition of the operating system.

Compatible whole-disk RHEL guest images should work without modification. To create your own custom disk image, see Creating images in the *Creating and Managing Images* guide.

4.4.1. Uploading the deploy images to the Image service

You must upload the deploy images installed by director to the Image service. The deploy image consists of the following two images:

- The kernel image: **/tftpboot/agent.kernel**
- The ramdisk image: **/tftpboot/agent.ramdisk**

These images are installed in the home directory. For more information on how the deploy images were installed, see Obtaining images for overcloud nodes .

Procedure

- Extract the images and upload them to the Image service:

```
$ 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
```

4.5. CONFIGURING DEPLOY INTERFACES

When you provision bare metal nodes, the Bare Metal Provisioning service (ironic) on the overcloud writes a base operating system image to the disk on the bare metal node. By default, the deploy interface mounts the image on an iSCSI mount and then copies the image to disk on each node. Alternatively, you can use direct deploy, which writes disk images from a HTTP location directly to disk on bare metal nodes.

NOTE

Support for the iSCSI deploy interface will be deprecated in Red Hat OpenStack Platform (RHOSP) version 17.0, and will be removed in RHOSP 18.0. Direct deploy will be the default deploy interface from RHOSP 17.0.

Deploy interfaces have a critical role in the provisioning process. Deploy interfaces orchestrate the deployment and define the mechanism for transferring the image to the target disk.

Prerequisites

- Dependent packages configured on the bare metal service nodes that run **ironic-conductor**.
- Configure OpenStack Compute (nova) to use the bare metal service endpoint.
- Create flavors for the available hardware, and nova must boot the new node from the correct flavor.
- Images must be available in the Image service (glance):
  - bm-deploy-kernel
  - bm-deploy-ramdisk
  - user-image
  - user-image-vmlinuz
  - user-image-initrd
- Hardware to enroll with the Ironic API service.

Workflow

Use the following example workflow to understand the standard deploy process. Depending on the ironic driver interfaces that you use, some of the steps might differ:

1. The Nova scheduler receives a boot instance request from the Nova API.
2. The Nova scheduler identifies the relevant hypervisor and identifies the target physical node.
3. The Nova compute manager claims the resources of the selected hypervisor.
4. The Nova compute manager creates unbound tenant virtual interfaces (VIFs) in the Networking service according to the network interfaces that the nova boot request specifies.
5. Nova compute invokes **driver.spawn** from the Nova compute virt layer to create a spawn task that contains all of the necessary information. During the spawn process, the virt driver completes the following steps.
   a. Updates the target ironic node with information about the deploy image, instance UUID, requested capabilities, and flavor properties.
   b. Calls the ironic API to validate the power and deploy interfaces of the target node.
   c. Attaches the VIFs to the node. Each neutron port can be attached to any ironic port or group. Port groups have higher priority than ports.
   d. Generates config drive.
6. The Nova ironic virt driver issues a deploy request with the Ironic API to the Ironic conductor that services the bare metal node.
7. Virtual interfaces are plugged in and the Neutron API updates DHCP to configure PXE/TFTP options.

8. The ironic node boot interface prepares (i)PXE configuration and caches the deploy kernel and ramdisk.
9. The ironic node management interface issues commands to enable network boot of the node.
10. The ironic node deploy interface caches the instance image, kernel, and ramdisk, if necessary.
11. The ironic node power interface instructs the node to power on.
12. The node boots the deploy ramdisk.
13. With iSCSI deployment, the conductor copies the image over iSCSI to the physical node. With direct deployment, the deploy ramdisk downloads the image from a temporary URL. This URL must be a Swift API compatible object store or a HTTP URL.
14. The node boot interface switches PXE configuration to refer to instance images and instructs the ramdisk agent to soft power off the node. If the soft power off fails, the bare metal node is powered off with IPMI/BMC.
15. The deploy interface instructs the network interface to remove any provisioning ports, binds the tenant ports to the node, and powers the node on.

The provisioning state of the new bare metal node is now **active**.

4.5.1. Configuring the direct deploy interface on the overcloud

The iSCSI deploy interface is the default deploy interface. However, you can enable the direct deploy interface to download an image from a HTTP location to the target disk.

NOTE

Support for the iSCSI deploy interface will be deprecated in Red Hat OpenStack Platform (RHOSP) version 17.0, and will be removed in RHOSP 18.0. Direct deploy will be the default deploy interface from RHOSP 17.0.

Prerequisites

- Your overcloud node memory **tmpfs** must have at least 8GB of RAM.

Procedure

1. Create or modify a custom environment file **/home/stack/templates/direct_deploy.yaml** and specify the **IronicEnabledDeployInterfaces** and the **IronicDefaultDeployInterface** parameters.

```
parameter_defaults:
  IronicEnabledDeployInterfaces: direct
  IronicDefaultDeployInterface: direct
```

   If you register your nodes with iSCSI, retain the **iscsi** value in the **IronicEnabledDeployInterfaces** parameter:

```
parameter_defaults:
  IronicEnabledDeployInterfaces: direct,iscsi
  IronicDefaultDeployInterface: direct
```

2. By default, the Bare Metal Provisioning service (ironic) agent on each node obtains the image stored in the Object Storage Service (swift) through a HTTP link. Alternatively, ironic can stream this image directly to the node through the **ironic-conductor** HTTP server. To change the service that provides the image, set the **IronicImageDownloadSource** to **http** in the **/home/stack/templates/direct_deploy.yaml** file:

```
parameter_defaults:
  IronicEnabledDeployInterfaces: direct
  IronicDefaultDeployInterface: direct
  IronicImageDownloadSource: http
```

3. Include the custom environment with your overcloud deployment:

```
$ openstack overcloud deploy \
  --templates \
  ...
  -e /usr/share/openstack-tripleo-heat-templates/environments/services/ironic.yaml \
  -e /home/stack/templates/direct_deploy.yaml \
  ...
```

Wait until deployment completes.

NOTE

If you did not specify **IronicDefaultDeployInterface** or want to use a different deploy interface, specify the deploy interface when you create or update a node:

```
$ openstack baremetal node create --driver ipmi --deploy-interface direct
$ openstack baremetal node set <NODE> --deploy-interface direct
```

4.6. ADDING PHYSICAL MACHINES AS BARE METAL NODES

Use one of the following methods to enroll a bare metal node:

- Prepare an inventory file with the node details, import the file into the Bare Metal Provisioning service, and make the nodes available.
- Register a physical machine as a bare metal node, and then manually add its hardware details and create ports for each of its Ethernet MAC addresses. You can perform these steps on any node that has your **overcloudrc** file.

4.6.1. Enrolling a bare metal node with an inventory file

Prepare an inventory file with the node details, import the file into the Bare Metal Provisioning service (ironic), and make the nodes available.

Prerequisites

- An overcloud deployment that includes the Bare Metal Provisioning service. For more information, see Deploying an overcloud with the Bare Metal Provisioning service .

Procedure

1. Create an inventory file, **overcloud-nodes.yaml**, that includes the node details. You can enroll multiple nodes with one file.

```
nodes:
    - name: node0
      driver: ipmi
      driver_info:
        ipmi_address: <ipmi_ip>
        ipmi_username: <user>
        ipmi_password: <password>
        [<property>: <value>]
      properties:
        cpus: <cpu_count>
        cpu_arch: <cpu_arch>
        memory_mb: <memory>
        local_gb: <root_disk>
        root_device:
            serial: <serial>
      ports:
        - address: <mac_address>
```

   - Replace **<ipmi_ip>** with the address of the Bare Metal controller.
   - Replace **<user>** with your username.
   - Replace **<password>** with your password.
   - Optional: Replace **<property>: <value>** with an IPMI property that you want to configure, and the property value. For information on the available properties, see Intelligent Platform Management Interface (IPMI) power management driver.
   - Replace **<cpu_count>** with the number of CPUs.
   - Replace **<cpu_arch>** with the type of architecture of the CPUs.
   - Replace **<memory>** with the amount of memory in MiB.
   - Replace **<root_disk>** with the size of the root disk in GiB. Only required when the machine has multiple disks.
   - Replace **<serial>** with the serial number of the disk that you want to use for deployment.
   - Replace **<mac_address>** with the MAC address of the NIC used to PXE boot.
   - --driver-info <property>=<value>

2. Source the **overcloudrc** file:

```
$ source ~/overcloudrc
```

3. Import the inventory file into the Bare Metal Provisioning service:

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

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

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

```
$ openstack baremetal node set <node> \
 --driver-info deploy_kernel=<kernel_file> \
 --driver-info deploy_ramdisk=<initramfs_file>
```

   - Replace **<node>** with the name or ID of the node.
   - Replace **<kernel_file>** with the path to the **.kernel** image, for example, **file:///var/lib/ironic/httpboot/agent.kernel**.
   - Replace **<initramfs_file>** with the path to the **.initramfs** image, for example, **file:///var/lib/ironic/httpboot/agent.ramdisk**.

5. Optional: Specify the IPMI cipher suite for each node:

```
$ openstack baremetal node set <node> \
 --driver-info ipmi_cipher_suite=<version>
```

   - Replace **<node>** with the name or ID of the node.
   - Replace **<version>** with the cipher suite version to use on the node. Set to one of the following valid values:
     - **3** - The node uses the AES-128 with SHA1 cipher suite.
     - **17** - The node uses the AES-128 with SHA256 cipher suite.

6. Set the provisioning state of the node to **available**:

```
$ openstack baremetal node manage <node>
$ openstack baremetal node provide <node>
```

   The Bare Metal Provisioning service cleans the node if you enabled node cleaning.

7. Set the local boot option on the node:

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

8. Check that the nodes are enrolled:

```
$ openstack baremetal node list
```

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

4.6.2. Enrolling a bare-metal node manually

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. You can perform these steps on any node that has your **overcloudrc** file.

Prerequisites

- An overcloud deployment that includes the Bare Metal Provisioning service. For more information, see Deploying an overcloud with the Bare Metal Provisioning service .

- The driver for the new node must be enabled by using the **IronicEnabledHardwareTypes** parameter. For more information about supported drivers, see Bare metal drivers.

Procedure

1. Log in to the undercloud host as the **stack** user.
2. Source the overcloud credentials file:

```
(undercloud)$ source ~/overcloudrc
```

3. Add a new node:

```
$ openstack baremetal node create --driver <driver_name> --name <node_name>
```

   - Replace **<driver_name>** with the name of the driver, for example, **ipmi**.
   - Replace **<node_name>** with the name of your new bare-metal node.

4. Note the UUID assigned to the node when it is created.
5. Set the boot option to **local** for each registered node:

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

   Replace **<node>** with the UUID of the bare metal node.

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

```
$ openstack baremetal node set <node> \
  --driver-info deploy_kernel=<kernel_file> \
  --driver-info deploy_ramdisk=<initramfs_file>
```

   - Replace **<node>** with the ID of the bare metal node.
   - Replace **<kernel_file>** with the path to the **.kernel** image, for example, **file:///var/lib/ironic/httpboot/agent.kernel**.
   - Replace **<initramfs_file>** with the path to the **.initramfs** image, for example, **file:///var/lib/ironic/httpboot/agent.ramdisk**.

7. Update the node properties to match the hardware specifications on the node:

```
$ openstack baremetal node set <node> \
  --property cpus=<cpu> \
  --property memory_mb=<ram> \
  --property local_gb=<disk> \
  --property cpu_arch=<arch>
```

   - Replace **<node>** with the ID of the bare metal node.
   - Replace **<cpu>** with the number of CPUs.
   - Replace **<ram>** with the RAM in MB.

- Replace **<disk>** with the disk size in GB.
- Replace **<arch>** with the architecture type.

8. Optional: Specify the IPMI cipher suite for each node:

```
$ openstack baremetal node set <node> \
 --driver-info ipmi_cipher_suite=<version>
```

- Replace **<node>** with the ID of the bare metal node.
- Replace **<version>** with the cipher suite version to use on the node. Set to one of the following valid values:
  - **3** - The node uses the AES-128 with SHA1 cipher suite.
  - **17** - The node uses the AES-128 with SHA256 cipher suite.

9. Optional: Specify the IPMI details for each node:

```
$ openstack baremetal node set <node> \
 --driver-info <property>=<value>
```

- Replace **<node>** with the ID of the bare metal node.
- Replace **<property>** with the IPMI property that you want to configure. For information on the available properties, see Intelligent Platform Management Interface (IPMI) power management driver.
- Replace **<value>** with the property value.

10. Optional: If you have multiple disks, set the root device hints to inform the deploy ramdisk which disk to use for deployment:

```
$ openstack baremetal node set <node> \
 --property root_device='{"<property>": "<value>"}'
```

- Replace **<node>** with the ID of the bare metal node.
- Replace **<property>** and **<value>** with details about the disk that you want to use for deployment, for example **root_device='{"size": "128"}'**
  RHOSP supports the following properties:
  - **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 Use this property only for devices with persistent names.

NOTE

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

11. Inform the Bare Metal Provisioning service of the node 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>** with the unique ID of the bare metal node.
- Replace **<mac_address>** with the MAC address of the NIC used to PXE boot.

12. Validate the configuration of the node:

```
$ openstack baremetal node validate <node>
+------------+--------+--------------------------------------------+
| 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   |                                            |
+------------+--------+--------------------------------------------+
```

The validation output **Result** indicates the following:

- **False**: The interface has failed validation. If the reason provided includes missing the **instance_info** parameters **[\'ramdisk', \'kernel', and \'image_source']**, this might be because the Compute service populates those missing parameters at the beginning of the deployment process, therefore they have not been set at this point. If you are using a whole disk image, then you might need to only set **image_source** to pass the validation.
- **True**: The interface has passed validation.

- **None**: The interface is not supported for your driver.

4.6.3. Bare-metal node provisioning states

A bare-metal node transitions through several provisioning states during its lifetime. API requests and conductor events performed on the node initiate the transitions. There are two categories of provisioning states: "stable" and "in transition".

Use the following table to understand the provisioning states a node can be in, and the actions that are available for you to use to transition the node from one provisioning state to another.

Table 4.1. Provisioning states

State	Category	Description
enroll	Stable	The initial state of each node. For information on enrolling a node, see Adding physical machines as bare metal nodes
verifying	In transition	The Bare Metal Provisioning service validates that it can manage the node by using the **driver_info** configuration provided during the node enrollment.
manageable	Stable	The node is transitioned to the manageable state when the Bare Metal Provisioning service has verified that it can manage the node. You can transition the node from the **manageable** state to one of the following states by using the following commands: • **openstack baremetal node adopt → adopting → active** • **openstack baremetal node provide → cleaning → available** • **openstack baremetal node clean → cleaning → available** • **openstack baremetal node inspect → inspecting → manageable** You must move a node to the **manageable** state after it is transitioned to one of the following failed states: • **adopt failed** • **clean failed** • **inspect failed** Move a node into the **manageable** state when you need to update the node.
inspecting	In transition	The Bare Metal Provisioning service uses node introspection to update the hardware-derived node properties to reflect the current state of the hardware. The node transitions to **manageable** for synchronous inspection, and **inspect wait** for asynchronous inspection. The node transitions to **inspect failed** if an error occurs.

State	Category	Description
inspect wait	In transition	The provision state that indicates that an asynchronous inspection is in progress. If the node inspection is successful, the node transitions to the **manageable** state.
inspect failed	Stable	The provisioning state that indicates that the node inspection failed. You can transition the node from the **inspect failed** state to one of the following states by using the following commands: • **openstack baremetal node inspect → inspecting → manageable** • **openstack baremetal node manage → manageable**
cleaning	In transition	Nodes in the **cleaning** state are being scrubbed and reprogrammed into a known configuration. When a node is in the **cleaning** state, depending on the network management, the conductor performs the following tasks: • Out-of-band: The conductor performs the **clean** step. • In-band: The conductor prepares the environment to boot the ramdisk for running the in-band clean steps. The preparation tasks include building the PXE configuration files, and configuring the DHCP.
clean wait	In transition	Nodes in the **clean wait** state are being scrubbed and reprogrammed into a known configuration. This state is similar to the **cleaning** state except that in the **clean wait** state, the conductor is waiting for the ramdisk to boot or the **clean** step to finish. You can interrupt the cleaning process of a node in the **clean wait** state by running **openstack baremetal node abort**.

State	Category	Description
available	Stable	After nodes have been successfully preconfigured and cleaned, they are moved into the **available** state and are ready to be provisioned. You can transition the node from the **available** state to one of the following states by using the following commands: • **openstack baremetal node deploy → deploying → active** • **openstack baremetal node manage → manageable**
deploying	In transition	Nodes in the **deploying** state are being prepared for a workload, which involves performing the following tasks: • Setting appropriate BIOS options for the node deployment. • Partitioning drives and creating file systems. • Creating any additional resources that may be required by additional subsystems, such as the node-specific network configuration, and a configuratin drive partition.
wait call-back	In transition	Nodes in the **wait call-back** state are being prepared for a workload. This state is similar to the **deploying** state except that in the **wait call-back** state, the conductor is waiting for a task to complete before preparing the node. For example, the following tasks must be completed before the conductor can prepare the node: • The ramdisk has booted. • The bootloader is installed. • The image is written to the disk. You can interrupt the deployment of a node in the **wait call-back** state by running **openstack baremetal node delete** or **openstack baremetal node undeploy**.

State	Category	Description
deploy failed	Stable	The provisioning state that indicates that the node deployment failed. You can transition the node from the **deploy failed** state to one of the following states by using the following commands: • **openstack baremetal node deploy → deploying → active** • **openstack baremetal node rebuild → deploying → active** • **openstack baremetal node delete → deleting → cleaning → clean wait → cleaning → available** • **openstack baremetal node undeploy → deleting → cleaning → clean wait → cleaning → available**
active	Stable	Nodes in the **active** state have a workload running on them. The Bare Metal Provisioning service may regularly collect out-of-band sensor information, including the power state. You can transition the node from the **active** state to one of the following states by using the following commands: • **openstack baremetal node delete → deleting → available** • **openstack baremetal node undeploy → cleaning → available** • **openstack baremetal node rebuild → deploying → active** • **openstack baremetal node rescue → rescuing → rescue**
deleting	In transition	When a node is in the **deleting** state, the Bare Metal Provisioning service disassembles the active workload and removes any configuration and resources it added to the node during the node deployment or rescue. Nodes transition quickly from the **deleting** state to the **cleaning** state, and then to the **clean wait** state.
error	Stable	If a node deletion is unsuccessful, the node is moved into the **error** state. You can transition the node from the**error** state to one of the following states by using the following commands: • **openstack baremetal node delete → deleting → available** • **openstack baremetal node undeploy → cleaning → available**

State	Category	Description
adopting	In transition	You can use the **openstack baremetal node adopt** command to transition a node with an existing workload directly from **manageable** to **active** state without first cleaning and deploying the node. When a node is in the **adopting** state the Bare Metal Provisioning service has taken over management of the node with its existing workload.
rescuing	In transition	Nodes in the **rescuing** state are being prepared to perform the following rescue operations: • Setting appropriate BIOS options for the node deployment. • Creating any additional resources that may be required by additional subsystems, such as node-specific network configurations.
rescue wait	In transition	Nodes in the **rescue wait** state are being rescued. This state is similar to the **rescuing** state except that in the **rescue wait** state, the conductor is waiting for the ramdisk to boot, or to execute parts of the rescue which need to run in-band on the node, such as setting the password for user named rescue. You can interrupt the rescue operation of a node in the **rescue wait** state by running **openstack baremetal node abort**.
rescue failed	Stable	The provisioning state that indicates that the node rescue failed. You can transition the node from the **rescue failed** state to one of the following states by using the following commands: • **openstack baremetal node rescue → rescuing → rescue** • **openstack baremetal node unrescue → unrescuing → active** • **openstack baremetal node delete → deleting → available**
rescue	Stable	Nodes in the **rescue** state are running a rescue ramdisk. The Bare Metal Provisioning service may regularly collect out-of-band sensor information, including the power state. You can transition the node from the **rescue** state to one of the following states by using the following commands: • **openstack baremetal node unrescue → unrescuing → active** • **openstack baremetal node delete → deleting → available**

State	Category	Description
unrescuing	In transition	Nodes in the **unrescuing** state are being prepared to transition from the **rescue** state to the **active** state.
unrescue failed	Stable	The provisioning state that indicates that the node unrescue operation failed. You can transition the node from the **unrescue failed** state to one of the following states by using the following commands: • **openstack baremetal node rescue → rescuing → rescue** • **openstack baremetal node unrescue → unrescuing → active** • **openstack baremetal node delete → deleting → available**

4.7. CONFIGURING REDFISH VIRTUAL MEDIA BOOT

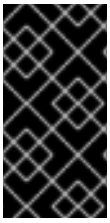
IMPORTANT

This feature 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.

You can use Redfish virtual media boot to supply a boot image to the Baseboard Management Controller (BMC) of a node so that the BMC can insert the image into one of the virtual drives. The node can then boot from the virtual drive into the operating system that exists in the image.

Redfish hardware types support booting deploy, rescue, and user images over virtual media. The Bare Metal Provisioning service (ironic) uses kernel and ramdisk images associated with a node to build bootable ISO images for UEFI or BIOS boot modes at the moment of node deployment. The major advantage of virtual media boot is that you can eliminate the TFTP image transfer phase of PXE and use HTTP GET, or other methods, instead.

4.7.1. Deploying a bare metal server with Redfish virtual media boot



IMPORTANT

This feature 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.

To boot a node with the **redfish** hardware type over virtual media, set the boot interface to **redfish-virtual-media** and, for UEFI nodes, define the EFI System Partition (ESP) image. Then configure an enrolled node to use Redfish virtual media boot.

Prerequisites

- Redfish driver enabled in the **enabled_hardware_types** parameter in the **undercloud.conf** file.
- A bare metal node registered and enrolled.
- IPA and instance images in the Image Service (glance).
- For UEFI nodes, you must also have an EFI system partition image (ESP) available in the Image Service (glance).
- A bare metal flavor.
- A network for cleaning and provisioning.
- Sushy library installed:

```
$ sudo yum install sushy
```

Procedure

1. Set the Bare Metal service (ironic) boot interface to **redfish-virtual-media**:

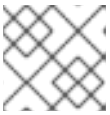
```
$ openstack baremetal node set --boot-interface redfish-virtual-media $NODE_NAME
```

   Replace **$NODE_NAME** with the name of the node.

2. For UEFI nodes, set the boot mode to **uefi**:

```
$ openstack baremetal node set --property capabilities="boot_mode:uefi" $NODE_NAME
```

   Replace **$NODE_NAME** with the name of the node.



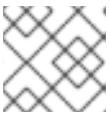
NOTE

For BIOS nodes, do not complete this step.

3. For UEFI nodes, define the EFI System Partition (ESP) image:

```
$ openstack baremetal node set --driver-info bootloader=$ESP $NODE_NAME
```

   Replace **$ESP** with the glance image UUID or URL for the ESP image, and replace **$NODE_NAME** with the name of the node.



NOTE

For BIOS nodes, do not complete this step.

4. Create a port on the bare metal node and associate the port with the MAC address of the NIC on the bare metal node:

```
$ openstack baremetal port create --pxe-enabled True --node $UUID $MAC_ADDRESS
```

Replace **$UUID** with the UUID of the bare metal node, and replace **$MAC_ADDRESS** with the MAC address of the NIC on the bare metal node.

5. Create the new bare metal server:

```
$ openstack server create \
    --flavor baremetal \
    --image $IMAGE \
    --network $NETWORK \
    test_instance
```

Replace **$IMAGE** and **$NETWORK** with the names of the image and network that you want to use.

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

OpenStack Compute uses host aggregates to partition availability zones, and group together nodes that have specific shared properties. When an instance is provisioned, the Compute 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.

Complete the steps in this section to perform the following operations:

- Add the property **baremetal** to your flavors and set 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.

Prerequisites

- A successful overcloud deployment that includes the Bare Metal Provisioning service. For more information, see Deploying an overcloud with the Bare Metal Provisioning service .

Procedure

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 that virtual instances use:

```
$ 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 you deployed the overcloud, add it now to the existing list under **scheduler_default_filters** in the _/**etc**/**nova**/**nova.conf**_ file:

```
AggregateInstanceExtraSpecsFilter
```

CHAPTER 5. ADMINISTERING BARE METAL NODES

After you deploy an overcloud that includes the Bare Metal Provisioning service (ironic), you can provision a physical machine on an enrolled bare metal node and launch bare metal instances in your overcloud.

Prerequisites

- A successful overcloud deployment that includes the Bare Metal Provisioning service. For more information, see Deploying an overcloud with the Bare Metal Provisioning service .

5.1. LAUNCHING BARE METAL INSTANCES

You can launch instances either from the command line or from the OpenStack dashboard.

Prerequisites

- A successful overcloud deployment that includes the Bare Metal Provisioning service. For more information, see Deploying an overcloud with the Bare Metal Provisioning service .

5.1.1. Launching instances with the command line interface

You can create a bare-metal instance by using the OpenStack Client CLI.

Prerequisites

- A successful overcloud deployment that includes the Bare Metal Provisioning service. For more information, see Deploying an overcloud with the Bare Metal Provisioning service .

Procedure

1. Configure the shell to access the Identity service (keystone) as the administrative user:

```
$ source ~/overcloudrc
```

2. Create your bare-metal instance:

```
$ openstack server create \
 --nic net-id=<network_uuid> \
 --flavor baremetal \
 --image <image_uuid> \
 myBareMetalInstance
```

   - Replace **<network_uuid>** with the unique identifier for the network that you created to use with the Bare Metal Provisioning service.
   - Replace **<image_uuid>** with the unique identifier for the image that has the software profile that your instance requires.

3. Check the status of the instance:

```
$ openstack server list --name myBareMetalInstance
```

5.1.2. Launching instances with the dashboard

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

Prerequisites

- A successful overcloud deployment that includes the Bare Metal Provisioning service. For more information, see Deploying an overcloud with the Bare Metal Provisioning service .

Procedure

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 image that you choose moves 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 that you created for the Bare Metal Provisioning service is selected here.
   - If you want 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**.

5.2. CONFIGURING 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 be used only 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 must have the same physical network. The Bare Metal Provisioning service uses **configdrive** to support configuration of port groups in the instances.

NOTE

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

- A successful overcloud deployment that includes the Bare Metal Provisioning service. For more information, see Deploying an overcloud with the Bare Metal Provisioning service .

5.2.1. Configuring port groups on switches manually

To configure port groups in a bare metal deployment, you must configure the port groups on the switches manually. You must ensure 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.

With port group fallback, all the ports in a port group can 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.

Prerequisites

- A successful overcloud deployment that includes the Bare Metal Provisioning service. For more information, see Deploying an overcloud with the Bare Metal Provisioning service .

5.2.2. Configuring port groups in the Bare Metal Provisioning service

Create a port group to aggregate multiple network interfaces into a single *bonded interface*.

Prerequisites

- A successful overcloud deployment that includes the Bare Metal Provisioning service. For more information, see Deploying an overcloud with the Bare Metal Provisioning service .

Procedure

1. Create a port group by specifying the node to which it belongs, its name, address, mode, properties and whether 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 use the **openstack baremetal port group set** command to update a port group.

   If you do not specify an address, the deployed instance port group address is the same as the OpenStack Networking port. If you do not attach the neutron port, the port group configuration fails.

   During interface attachment, port groups have a higher priority than the ports, so they are 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 are ignored.

NOTE

You must configure port groups manually in standalone mode either in the image or by generating the **configdrive** and adding it to the node's **instance_info**. Ensure that 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 is configured properly, run the following command:

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

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

5.3. DETERMINING THE HOST TO IP ADDRESS MAPPING

Use the following commands to determine which IP addresses are assigned to which host and bare metal node. With these commands, you can view the host to IP mapping from the undercloud without accessing the hosts directly.

Prerequisites

- A successful overcloud deployment that includes the Bare Metal Provisioning service. For more information, see Deploying an overcloud with the Bare Metal Provisioning service .

Procedure

1. Run the following command to display the IP address for each host:

```
(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                |
+--------------+---------------------------------------------------------------+
```

2. To filter a particular host, run the following command:

```
(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
```

3. To map the hosts to bare metal nodes, run the following command:

```
(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"
  }
]
```

5.4. ATTACHING AND DETACHING VIRTUAL NETWORK INTERFACES

The Bare Metal Provisioning service has an API that you can use to manage the mapping between virtual network interfaces. For example, the interfaces in the OpenStack Networking service and your physical interfaces (NICs). You can configure these interfaces for each Bare Metal Provisioning node to set the virtual network interface (VIF) to physical network interface (PIF) mapping logic. To configure the interfaces, use the **openstack baremetal node vif*** commands.

Prerequisites

- A successful overcloud deployment that includes the Bare Metal Provisioning service. For more information, see Deploying an overcloud with the Bare Metal Provisioning service .

Procedure

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 Provisioning service updates the virtual port in the OpenStack Networking service with the actual MAC address of the physical port. Check this port address:

```
$ 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 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 that 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 Provisioning 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 that it recognizes the new IP address:

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

After you detach or attach interfaces, the bare metal OS removes, adds, or modifies the network interfaces that have changed. When you replace a port, a DHCP request obtains the new IP address, but this might take some time because the old DHCP lease is still valid. To initiate these changes immediately, reboot the bare metal host.

5.5. CONFIGURING NOTIFICATIONS FOR THE BARE METAL PROVISIONING SERVICE

You can configure the Bare Metal Provisioning service (ironic) to display notifications for different events that occur within the service. External services can use these notifications for billing purposes, monitoring a data store, and other purposes. To enable notifications for the Bare Metal Provisioning service, you must set the following options in your **ironic.conf** configuration file.

Prerequisites

- A successful overcloud deployment that includes the Bare Metal Provisioning service. For more information, see Deploying an overcloud with the Bare Metal Provisioning service .

Procedure

- The **notification_level** option in the **[DEFAULT]** section determines the minimum priority level for which notifications are sent. You can set the values for this option 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 are 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 that describes the contents of the message.

5.6. CONFIGURING AUTOMATIC POWER FAULT RECOVERY

The Bare Metal Provisioning service (ironic) has a string field **fault** that records power, cleaning, and rescue abort failures for nodes.

Table 5.1. Ironic node faults

Fault	Description
power failure	The node is in maintenance mode due to power sync failures that exceed the maximum number of retries.
clean failure	The node is in maintenance mode due to the failure of a cleaning operation.
rescue abort failure	The node is in maintenance mode due to the failure of a cleaning operation during rescue abort.
none	There is no fault present.

Conductor checks the value of this field periodically. If the conductor detects a **power failure** state and can successfully restore power to the node, the node is removed from maintenance mode and restored to operation.

NOTE

If the operator places a node in maintenance mode manually, the conductor does not automatically remove the node from maintenance mode.

The default interval is 300 seconds, however, you can configure this interval with director using hieradata.

Prerequisites

Prerequisites

- A successful overcloud deployment that includes the Bare Metal Provisioning service. For more information, see Deploying an overcloud with the Bare Metal Provisioning service .

Procedure

- Include the following hieradata to configure a custom recovery interval:

```
ironic::conductor::power_failure_recovery_interval
```

  To disable automatic power fault recovery, set the value to **0**.

5.7. INTROSPECTING OVERCLOUD NODES

Perform introspection of overcloud nodes to identify and store the specification of your nodes in director.

Procedure

1. Log in to the undercloud host as the **stack** user.
2. Source the **overcloudrc** credentials file:

```
$ source ~/overcloudrc
```

3. Run the introspection command:

```
$ openstack baremetal introspection start [--wait] <NODENAME>
```

   Replace <NODENAME> with the name or UUID of the node that you want to inspect.

4. Check the introspection status:

```
$ openstack baremetal introspection status <NODENAME>
```

   Replace <NODENAME> with the name or UUID of the node.

Next steps

- Extract introspection data:

```
$ openstack baremetal introspection data save <NODE-UUID>
```

  Replace <NODENAME> with the name or UUID of the node.

CHAPTER 6. BOOTING FROM CINDER VOLUMES

You can create volumes in the Block Storage service (cinder) and connect these volumes to bare metal instances that you create with the Bare Metal Provisioning service (ironic).

6.1. CINDER VOLUME BOOT FOR BARE METAL NODES

You can boot bare metal nodes from a block storage device that is stored in OpenStack Block Storage (cinder). OpenStack Bare Metal (ironic) connects bare metal nodes to volumes through an iSCSI interface.

Ironic enables this feature during the overcloud deployment. However, consider the following conditions before you deploy the overcloud:

- The overcloud requires the cinder iSCSI backend to be enabled. Set the **CinderEnableIscsiBackend** heat parameter to **true** during overcloud deployment.
- You cannot use the cinder volume boot feature with a Red Hat Ceph Storage backend.
- You must set the **rd.iscsi.firmware=1** kernel parameter on the boot disk.

6.2. CONFIGURING NODES FOR CINDER VOLUME BOOT

You must configure certain options for each bare metal node to successfully boot from a cinder volume.

Procedure

1. Log in to the undercloud as the **stack** user.
2. Source the overcloud credentials:

```
$ source ~/overcloudrc
```

3. Set the **iscsi_boot** capability to **true** and the **storage-interface** to **cinder** for the selected node:

```
$ openstack baremetal node set --property capabilities=iscsi_boot:true --storage-interface
cinder <NODEID>
```

   Replace **<NODEID>** with the ID of the chosen node.

4. Create an iSCSI connector for the node:

```
$ openstack baremetal volume connector create --node <NODEID> --type iqn --connector-id
iqn.2010-10.org.openstack.node<NUM>
```

   The connector ID for each node must be unique. In this example, the connector is **iqn.2010-10.org.openstack.node<NUM>** where **<NUM>** is an incremented number for each node.

6.3. CONFIGURING ISCSI KERNEL PARAMETERS ON THE BOOT DISK

You must enable the iSCSI booting in the kernel on the image. To accomplish this, mount the QCOW2 image and enable iSCSI components on the image.

Prerequisites

1. Download a Red Hat Enterprise Linux QCOW2 image and copy it to the **/home/stack/** directory on the undercloud. You can download Red Hat Enterprise Linux KVM images in QCOW2 format from the following pages:
   - Red Hat Enterprise Linux 7
   - Red Hat Enterprise Linux 8

Procedure

1. Log in to the undercloud as the **stack** user.
2. Mount the QCOW2 image and access it as the **root** user:
   a. Load the **nbd** kernel module:

   ```
   $ sudo modprobe nbd
   ```

   b. Connect the QCOW image as **/dev/nbd0**:

   ```
   $ sudo qemu-nbd --connect=/dev/nbd0 <IMAGE>
   ```

   c. Check the partitions on the NBD:

   ```
   $ sudo fdisk /dev/nbd0 -l
   ```

   New Red Hat Enterprise Linux QCOW2 images contain only one partition, which is usually named **/dev/nbd0p1** on the NBD.

   d. Create a mount point for the image:

   ```
   mkdir /tmp/mountpoint
   ```

   e. Mount the image:

   ```
   sudo mount /dev/nbd0p1 /tmp/mountpoint/
   ```

   f. Mount your **dev** directory so that the image has access to device information on the host:

   ```
   sudo mount -o bind /dev /tmp/mountpoint/dev
   ```

   g. Change the root directory to the mount point:

   ```
   sudo chroot /tmp/mountpoint /bin/bash
   ```

3. Configure iSCSI on the image:

NOTE

Some commands in this step might report the following error:

```
lscpu: cannot open /proc/cpuinfo: No such file or directory
```

This error is not critical and you can ignore the error.

a. Move the **resolv.conf** file to a temporary location:

```
# mv /etc/resolv.conf /etc/resolv.conf.bak
```

b. Create a temporary **resolv.conf** file to resolve DNS requests for the Red Hat Content Delivery Network. This example uses **8.8.8.8** for the nameserver:

```
# echo "nameserver 8.8.8.8" > /etc/resolv.conf
```

c. Register the mounted image to the Red Hat Content Delivery Network:

```
# subscription-manager register
```

Enter your user name and password when the command prompts you.

d. Attach a subscription that contains Red Hat Enterprise Linux:

```
# subscription-manager list --all --available
# subscription-manager attach --pool <POOLID>
```

Substitute **<POOLID>** with the pool ID of the subscription.

e. Disable the default repositories:

```
# subscription-manager repos --disable "*"
```

f. Enable the Red Hat Enterprise Linux repository:

- Red Hat Enterprise Linux 7:

```
# subscription-manager repos --enable "rhel-7-server-rpms"
```

- Red Hat Enterprise Linux 8:

```
# subscription-manager repos --enable "rhel-8-for-x86_64-baseos-eus-rpms"
```

g. Install the **iscsi-initiator-utils** package:

```
# yum install -y iscsi-initiator-utils
```

h. Unregister the mounted image:

```
# subscription-manager unregister
```

i. Restore the original **resolv.conf** file:

```
# mv /etc/resolv.conf.bak /etc/resolv.conf
```

j. Check the kernel version on the mounted image:

```
# rpm -qa kernel
```

For example, if the output is **kernel-3.10.0-1062.el7.x86_64**, the kernel version is **3.10.0-1062.el7.x86_64**. Note this kernel version for the next step.

NOTE

New Red Hat Enterprise Linux QCOW2 images have only one kernel version installed. If more than one kernel version is installed, use the latest one.

k. Add the **network** and **iscsi** dracut modules to the initramfs image:

```
# dracut --force --add "network iscsi" /boot/initramfs-<KERNELVERSION>.img
<KERNELVERSION>
```

Replace **<KERNELVERSION>** with the version number that you obtained from **rpm -qa kernel**. The following example uses **3.10.0-1062.el7.x86_64** as the kernel version:

```
# dracut --force --add "network iscsi" /boot/initramfs-3.10.0-1062.el7.x86_64.img 3.10.0-
1062.el7.x86_64
```

l. Exit from the mounted image back to your host operating system:

```
# exit
```

4. Unmount the image:

a. Unmount the **dev** directory from the temporary mount point:

```
$ sudo umount /tmp/mountpoint/dev
```

b. Unmount the image from the mount point:

```
$ sudo umount /tmp/mountpoint
```

c. Disconnect the QCOW2 image from **/dev/nbd0/**:

```
$ sudo qemu-nbd --disconnect /dev/nbd0
```

5. Rebuild the **grub** menu configuration on the image:

a. Install the **libguestfs-tools** package:

```
$ sudo yum -y install libguestfs-tools
```

IMPORTANT

If you install the **libguestfs-tools** package on the undercloud, disable **iscsid.socket** to avoid port conflicts with the **tripleo_iscsid** service on the undercloud:

```
$ sudo systemctl disable --now iscsid.socket
```

b. Set the **libguestfs** backend to use QEMU directly:

```
$ export LIBGUESTFS_BACKEND=direct
```

c. Update the grub configuration on the image:

```
$ guestfish -a /tmp/images/{{ dib_image }} -m /dev/sda3 sh "mount /dev/sda2 /boot/efi &&
rm /boot/grub2/grubenv && /sbin/grub2-mkconfig -o /boot/grub2/grub.cfg && cp
/boot/grub2/grub.cfg /boot/efi/EFI/redhat/grub.cfg && grubby --update-kernel=ALL --
args=\"rd.iscsi.firmware=1\" && cp /boot/grub2/grubenv /boot/efi/EFI/redhat/grubenv &&
echo Success"
```

6.4. CREATING AND USING A BOOT VOLUME IN CINDER

You must upload the iSCSI-enabled image to OpenStack Image Storage (glance) and create the boot volume in OpenStack Block Storage (cinder).

Procedure

1. Log in to the undercloud as the **stack** user.
2. Upload the iSCSI-enabled image to glance:

```
$ openstack image create --disk-format qcow2 --container-format bare --file rhel-server-7.7-
x86_64-kvm.qcow2 rhel-server-7.7-iscsi
```

3. Create a volume from the image:

```
$ openstack volume create --size 10 --image rhel-server-7.7-iscsi --bootable rhel-test-volume
```

4. Create a bare metal instance that uses the boot volume in cinder:

```
$ openstack server create --flavor baremetal --volume rhel-test-volume --key default rhel-test
```

CHAPTER 7. ML2 NETWORKING-ANSIBLE

You can enable and configure the **networking-ansible** ML2 driver on an overcloud with the Networking service (neutron) and integrate it with the Bare Metal Provisioning service (ironic)

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 any bare metal guest that uses a VLAN neutron network causes this driver to configure the physical switch using Ansible Engine Networking.

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

- Define a VLAN on the switch when creating a network in Red Hat OpenStack Platform (RHOSP)
- Assign a VLAN to an access port on the switch when creating or updating a port in RHOSP
- Remove a VLAN from an access port on the switch when deleting a port in RHOSP

7.2. NETWORKING REQUIREMENTS FOR NETWORKING-ANSIBLE

To enable **networking-ansible** functionality, your environment must include the following networking configuration:

- A network switch with Ansible Network Automation support:
  - Juniper Networks (**junos**)
  - Arista Extensible Operating System (**eos**)

IMPORTANT

Arista Extensible Operating System (**eos**) support 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.

- The network switch requires an SSH user so that Ansible Network Automation can interact with the device. This user requires the following 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.
- Prepare the VLANs that you want the switch to use. To prepare the VLANs, create each VLAN on the switch, and then delete each VLAN.

- 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 Bare Metal Provisioning service (ironic) 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 information about configuring the provisioning network, see Deploying an overcloud with the Bare Metal Provisioning 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
```

- The overcloud uses the introspection service to automatically identify certain hardware details and map them for other services to use. It is recommended that you enable the ironic introspection service to help map your interface-to-port details for **networking-ansible** to use. You can also accomplish this task manually.

For more information about deploying the Bare Metal Provisioning service (ironic), see Deploying an overcloud with the Bare Metal Provisioning service.

7.4. ENABLING NETWORKING-ANSIBLE ML2 FUNCTIONALITY

To enable the **networking-ansible** ML2 driver in your overcloud, you must add 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

A file that 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. The **ML2HostConfigs** parameter requires a **dict** value with details about your switches. Each initial key in the **dict** is a name for the switch. This value defines a specific **ansible: [switchname]** section in your OpenStack Networking (neutron) ML2 configuration. Each switch name key requires its own **dict** that contains the actual switch details. For example, to configure three switches, add three switch keys:

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

3. Each switch requires certain key value pairs in the **dict**:

ansible_network_os

(Required) The operating system of the switch. Options include **junos** and **eos**.

IMPORTANT

Arista Extensible Operating System (**eos**) support 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.

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. This is a required value when using introspection to perform automatic port 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: juno
      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 you run 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. The following example demonstrates how to include these files:

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

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 you deploy the overcloud with bare metal provisioning and the **networking-ansible** driver enabled, you must create provisioning and tenant networks for your bare metal nodes. You must also configure ports for your bare metal nodes either in access mode or trunk mode, depending on your requirements.

Access mode

In access mode, switch ports carry the traffic of only one VLAN and operate on a single broadcast domain. All traffic that arrives to access ports belongs to the VLAN that is assigned to the port.

Trunk mode

In trunk mode, switch ports can belong to more than one VLAN. You can use switch ports in trunk mode to carry the traffic of a group of VLANs, or if you want to exchange traffic between multiple switches with more than one VLAN.

IMPORTANT

This feature 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.

The Bare Metal service (ironic) uses **networking-ansible** to assign the switchport of the bare metal guest to the ironic provisioning network so that the provisioning process can complete successfully. After provisioning is complete, ironic assigns the switchport of the bare metal guest to the VLAN that the Networking service (neutron) assigns to the tenant networks of the bare metal guest.

7.5.1. Configuring networks for networking-ansible in access mode

After you deploy 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. For more information about configuring the provisioning network, see Configuring the Bare Metal Provisioning service after deployment .
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.

7.5.2. Configuring ports for bare metal guests in access mode

Bare metal guests require port information to connect to the switch. There are two methods to accomplish this:

- **Automatic:** Introspection of nodes. To use the automatic method, set the **mac** value for the respective switch as a part of the **ML2HostConfigs** parameter.
- **Manual:** Set the OpenStack Networking (neutron) port configuration. Use this method if your overcloud does not include bare metal introspection functionality.

Procedure

- **Automatic:**
  a. Run the introspection command:

```
$ openstack baremetal introspection start [--wait] <NODENAME>
```

The bare metal nodes obtain the MAC address of the switch during introspection. The **networking-ansible** ML2 driver uses this MAC address to map to the same MAC address that you define with the **mac** parameter for the respective switch in the **ML2HostConfigs** parameter.

  b. Wait until the introspection completes.

- **Manual:**

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=[SWITCH 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 that is 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 that connects to the bare metal node.

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

The name of the switch that connects to the bare metal node. The switch name must match the respective switch name that you define 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**.

7.5.3. Configuring networks for networking-ansible in trunk mode

IMPORTANT

This feature 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.

After you deploy 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

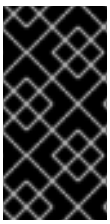
1. Create the provisioning network and subnet. This depends on the type of provisioning network you are using. For more information about configuring the provisioning network, see Configuring the Bare Metal Provisioning service after deployment .

2. Create a primary tenant VLAN network, a secondary tenant network, and subnets for each network that use the physical network that the guest is attached to:

```
$ openstack network create --provider-network-type vlan --provider-physical-network
baremetal primary-tenant-net
$ openstack network create --provider-network-type vlan --provider-physical-network
baremetal secondary-tenant-net
$ openstack subnet create --network primary-tenant-net --subnet-range 192.168.3.0/24 --
allocation-pool start=192.168.3.10,end=192.168.3.20 primary-tenant-subnet
$ openstack subnet create --network secondary-tenant-net --subnet-range 192.168.7.0/24 --
allocation-pool start=192.168.7.10,end=192.168.7.20 secondary-tenant-subnet
```

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

7.5.4. Configuring ports for bare metal guests in trunk mode



IMPORTANT

This feature 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.

Bare metal guests require port information to connect to the switch so that you can use the Bare Metal Provisioning service (ironic) to deploy on multiple networks with a single switch port. The switch port is configured in trunk mode using the VLANs that the Networking service (neutron) assigns from the supplied networks.

Complete the following steps to configure trunk ports for bare metal guests.

Procedure

1. Create a port and a trunk, and assign the port to the trunk as the parent port:

```
$ port create --network primary-tenant-net primary-port
$ network trunk create --parent-port primary-port my-trunk
```

2. Create a port for the secondary network and add the new port as a subport to the trunk:

```
$ port create --network secondary-tenant-net secondary-port
$ network trunk set --subport port=secondary-port,segmentation-type=vlan,segmentation-
id=1234 my-trunk
```

7.6. TESTING NETWORKING-ANSIBLE ML2 FUNCTIONS

After the **networking-ansible** configuration for the bare metal node is complete, create a bare metal workload to verify that the configuration is correct.

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.
- Configuration of the respective bare metal port used for the node connection on the switch. This port can be either an access port or a trunk port.
- 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.
- Disk images and key pairs available in the overcloud.

Procedure

1. Create the bare metal system:
   - To create a bare metal system that uses an access port, run the following command:

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

   - To create a bare metal system that uses a trunk port, run the following command:

     ```
     openstack server create --flavor baremetal --image overcloud-full --port {primary-port-uuid} --key default 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 PROVISIONING SERVICE

Diagnose issues in an environment that includes the Bare Metal Provisioning service (ironic).

8.1. PXE BOOT ERRORS

Use the following troubleshooting procedures to assess and remedy issues you might encounter with PXE boot.

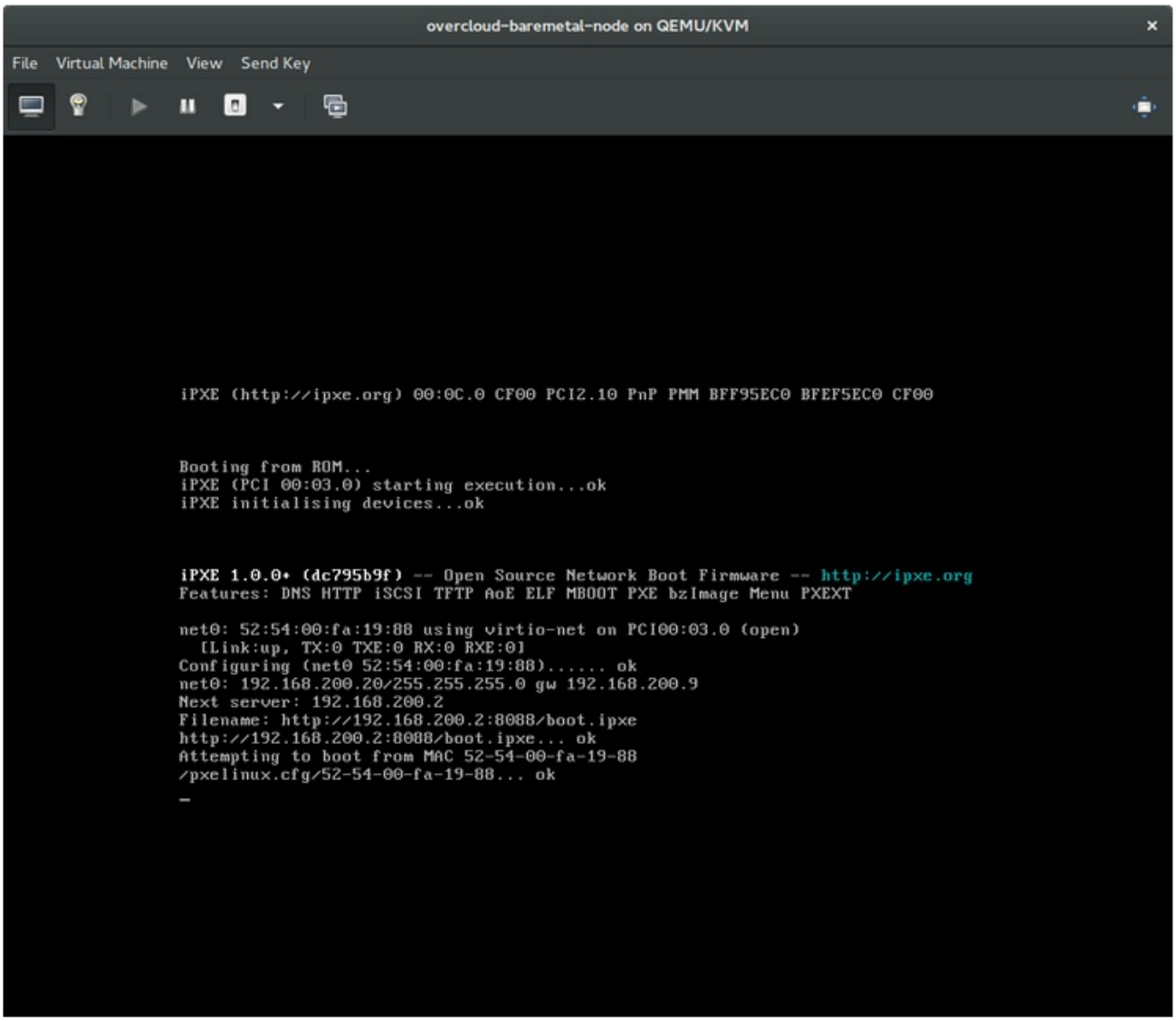
Permission Denied errors

If the console of your bare metal node returns a **Permission Denied** error, ensure that you have applied the appropriate SELinux context to the **/httpboot** and **/tftpboot** directories:

```
# 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 receive an IP address but then the process stops, you might be using the wrong PXE boot template in your **ironic.conf** file.



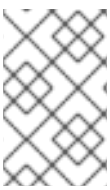

```
$ grep ^pxe_config_template ironic.conf
pxe_config_template=$pybasedir/drivers/modules/ipxe_config.template
```

The default template is **pxe_config.template**, so it is easy to omit the *i* and inadvertently enter **ipxe_config.template** instead.

8.2. LOGIN ERRORS AFTER THE BARE METAL NODE BOOTS

Failure to log in to the node when you use the root password that you set during configuration indicates that you are not booted into the deployed image. You might be logged in to the **deploy-kernel**/**deploy-ramdisk** image and the system has not yet loaded the correct image.

To fix this issue, verify that the PXE Boot Configuration file in the **/httpboot/pxelinux.cfg/MAC_ADDRESS** on the Compute or Bare Metal Provisioning 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 that the Bare Metal Provisioning service node uses is the Bare Metal network. If one of the endpoints is not on the network, the endpoint cannot reach the Bare Metal Provisioning 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=ipmi 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**.
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.
ipmi	The IPMI Driver in use by the Bare Metal Provisioning 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 Provisioning 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 PROVISIONING SERVICE DOES NOT RECEIVE THE CORRECT HOST NAME

If the Bare Metal Provisioning service does not receive the right host name, it means that **cloud-init** is failing. To fix this, connect the Bare Metal subnet to a router in the OpenStack Networking service. This configuration routes requests to the meta-data agent correctly.

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

If you cannot authenticate to the Identity service, check the **identity_uri** parameter in the **ironic.conf** file and ensure that you remove the **/v2.0** from the **keystone** AdminURL. For example, set the **identity_uri** to **http://IP:PORT**.

8.6. HARDWARE ENROLMENT

Incorrect node registration details can cause issues with enrolled hardware. Ensure that you enter property names and values correctly. When you input property names incorrectly, the system adds the properties to the node details but ignores them.

Use the **openstack baremetal node set** command to update node details. For example, update the amount of memory that the node is registered to use to 2 GB:

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

8.7. TROUBLESHOOTING IDRAC ISSUES

Redfish management interface fails to set boot device

When you use the **idrac-redfish** management interface with certain iDRAC firmware versions and attempt to set the boot device on a bare metal server with UEFI boot, iDRAC returns the following error:

```
Unable to Process the request because the value entered for the
parameter Continuous is not supported by the implementation.
```

If you encounter this issue, set the **force_persistent_boot_device** parameter in the **driver-info** on the node to **Never**:

```
openstack baremetal node set --driver-info force_persistent_boot_device=Never ${node_uuid}
```

Timeout when powering off

Some servers can be too slow when powering off, and time out. The default retry count is **6**, which results in a 30 second timeout. To increase the timeout duration to 90 seconds, set the

ironic::agent::rpc_response_timeout value to **18** in the undercloud hieradata overrides file and re-run the **openstack undercloud install** command:

```
ironic::agent::rpc_response_timeout: 18
```

Vendor passthrough timeout

When iDRAC is not available to execute vendor passthrough commands, these commands take too long and time out:

```
openstack baremetal node passthru call --http-method GET \
aed58dca-1b25-409a-a32f-3a817d59e1e0 list_unfinished_jobs
Timed out waiting for a reply to message ID 547ce7995342418c99ef1ea4a0054572 (HTTP 500)
```

To increase the timeout duration for messaging, increase the value of the **ironic::default::rpc_response_timeout** parameter in the undercloud hieradata overrides file and re-run the **openstack undercloud install** command:

```
ironic::default::rpc_response_timeout: 600
```

8.8. CONFIGURING THE SERVER CONSOLE

Console output from overcloud nodes is not always sent to the server console. If you want to view this output in the server console, you must configure the overcloud to use the correct console for your hardware. Use one of the following methods to perform this configuration:

- Modify the **KernelArgs** heat parameter for each overcloud role.
- Customize the **overcloud-full.qcow2** image that director uses to provision the overcloud nodes.

Prerequisites

- A successful undercloud installation. For more information, see the Director Installation and Usage guide.
- Overcloud nodes ready for deployment.

Modifying KernelArgs with heat during deployment

1. Log in to the undercloud host as the **stack** user.
2. Source the **stackrc** credentials file:

```
$ source stackrc
```

3. Create an environment file **overcloud-console.yaml** with the following content:

```
parameter_defaults:
  <role>Parameters:
    KernelArgs: "console=<console-name>"
```

Replace **<role>** with the name of the overcloud role that you want to configure, and replace **<console-name>** with the ID of the console that you want to use. For example, use the following snippet to configure all overcloud nodes in the default roles to use **tty0**:

```
parameter_defaults:
  ControllerParameters:
    KernelArgs: "console=tty0"
  ComputeParameters:
    KernelArgs: "console=tty0"
  BlockStorageParameters:
    KernelArgs: "console=tty0"
  ObjectStorageParameters:
    KernelArgs: "console=tty0"
  CephStorageParameters:
    KernelArgs: "console=tty0"
```

4. Include the **overcloud-console-tty0.yaml** file in your deployment command with the **-e** option.

Modifying the overcloud-full.qcow2 image

1. Log in to the undercloud host as the **stack** user.
2. Source the **stackrc** credentials file:

```
$ source stackrc
```

3. Modify the kernel arguments in the **overcloud-full.qcow2** image to set the correct console for your hardware. For example, set the console to **tty0**:

```
$ virt-customize --selinux-relabel -a overcloud-full.qcow2 --run-command 'grubby --update-kernel=ALL --args="console=tty0"'
```

4. Import the image into director:

```
$ openstack overcloud image upload --image-path /home/stack/images/overcloud-full.qcow2
```

5. Deploy the overcloud.

Verification

1. Log in to an overcloud node from the undercloud:

```
$ ssh heat-admin@<IP-address>
```

Replace **<IP-address>** with the IP address of an overcloud node.

2. Inspect the contents of the **/proc/cmdline** file and ensure that **console=** parameter is set to the value of the console that you want to use:

```
[heat-admin@controller-0 ~]$ cat /proc/cmdline
BOOT_IMAGE=(hd0,msdos2)/boot/vmlinuz-4.18.0-193.29.1.el8_2.x86_64
root=UUID=0ec3dea5-f293-4729-b676-5d38a611ce81 ro console=tty0
console=ttyS0,115200n81 no_timer_check crashkernel=auto rhgb quiet
```

CHAPTER 9. BARE METAL DRIVERS

You can configure bare metal nodes to use one of the drivers that are enabled in the Bare Metal Provisioning service. Each driver includes 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 by configuring the **IronicEnabledHardwareTypes** parameter in your **ironic.yaml** file. By default, **ipmi** and **redfish** are enabled.

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

9.1. INTELLIGENT PLATFORM MANAGEMENT INTERFACE (IPMI) POWER MANAGEMENT DRIVER

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 Provisioning service nodes require an IPMI that is connected to the shared Bare Metal network. IPMI power manager driver uses the **ipmitool** utility to remotely manage hardware. You can use the following **driver_info** properties to configure the IPMI power manager driver for a node:

Table 9.1. IPMI driver_info properties

Property	Description	Equivalent **ipmitool** option
ipmi_address	(Mandatory) The IP address or hostname of the node.	**-H**
ipmi_username	The IPMI user name.	**-U**
ipmi_password	The IPMI password. The password is written to a temporary file. You pass the filename to the **ipmitool** by using the **-f** option.	**-f**
ipmi_hex_kg_key	The hexadecimal Kg key for IPMIv2 authentication.	**-y**
ipmi_port	The remote IPMI RMCP port.	**-p**
ipmi_priv_level	IPMI privilege level. Set to one of the following valid values: • **ADMINISTRATOR** (default) • **CALLBACK** • **OPERATOR** • **USER**	**-L**

Property	Description	Equivalent **ipmitool** option
ipmi_protocol_version	The version of the IPMI protocol. Set to one of the following valid values: • **1.5** for lan • **2.0** for lanplus (default)	**-I**
ipmi_bridging	The type of bridging. Use with nested chassis management controllers (CMCs). Set to one of the following valid values: • **single** • **dual** • **no** (default)	n/a
ipmi_target_channel	Destination channel for a bridged request. Required only if **ipmi_bridging** is set to **single** or **dual**.	**-b**
ipmi_target_address	Destination address for a bridged request. Required only if **ipmi_bridging** is set to **single** or **dual**.	**-t**
ipmi_transit_channel	Transit channel for a bridged request. Required only if **ipmi_bridging** is set to **dual**.	**-B**
ipmi_transit_address	Transit address for bridged request. Required only if **ipmi_bridging** is set to **dual**.	**-T**
ipmi_local_address	Local IPMB address for bridged requests. Use only if **ipmi_bridging** is set to **single** or **dual**.	**-m**
ipmi_force_boot_device	Set to **true** to specify if the Bare Metal Provisioning service should specify the boot device to the BMC each time the server is turned on. The BMC is not capable of remembering the selected boot device across power cycles. Disabled by default.	n/a
ipmi_disable_boot_timeout	Set to **false** to not send a raw IPMI command to disable the 60 second timeout for booting on the node.	n/a
ipmi_cipher_suite	The IPMI cipher suite version to use on the node. Set to one of the following valid values: • **3** for AES-128 with SHA1 • **17** for AES-128 with SHA256	n/a

9.2. REDFISH

A standard RESTful API for IT infrastructure developed by the Distributed Management Task Force (DMTF). You can use the following **driver_info** properties to configure the Bare Metal Provisioning serive (ironic) connection to Redfish:

Table 9.2. Redfish driver_info properties

Property	Description
redfish_address	(Mandatory) The IP address of the Redfish controller. The address must include the authority portion of the URL. If you do not include the scheme it defaults to **https**.
redfish_system_id	The canonical path to the system resource the Redfish driver interacts with. The path must include the root service, version, and the unique path to the system within the same authority as the **redfish_address** property. For example: **/redfish/v1/Systems/CX34R87**. This property is only required if the target BMC manages more than one resource.
redfish_username	The Redfish username.
redfish_password	The Redfish password.
redfish_verify_ca	Either a Boolean value, a path to a CA_BUNDLE file, or a directory with certificates of trusted CAs. If you set this value to **True** the driver verifies the host certificates. If you set this value to **False** the driver ignores verifying the SSL certificate. If you set this value to a path, the driver uses the specified certificate or one of the certificates in the directory. The default is **True**.
redfish_auth_type	The Redfish HTTP client authentication method. Set to one of the following valid values: • **basic** • **session** (recommended) • **auto** (default) - Uses the **session** authentication method when available, and the **basic** authentication method when the **session** method is not available.

9.3. 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 Provisioning service nodes require a DRAC that is connected to the shared Bare Metal Provisioning network. Enable the **idrac** driver, and set the following information in the **driver_info** of the node:

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

- Optional: **drac_port** - The port to use for the WS-Management endpoint. The default is port **443**.
- Optional: **drac_path** - The path to use for the WS-Management endpoint. The default path is /**wsman**.
- Optional: **drac_protocol** - The protocol to use for the WS-Management endpoint. Valid values: **http**, **https**. The default protocol is **https**.

9.4. 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 Provisioning service node, the node requires an iRMC interface that is connected to the shared Bare Metal network. Enable the **irmc** driver, and set the following information in the **driver_info** of the node:

- **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 the **ironic.conf** file:

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

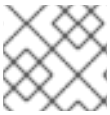
   Replace *METHOD* with **scci** or **ipmitool**.

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

```
# dnf 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.

9.5. 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 **ilo** driver, and set the following information in the **driver_info** of the node:

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

- **ilo_password** - The iLO password.