Red Hat OpenStack Platform 17.0-Beta

Bare Metal Provisioning

Install and configure the Bare Metal Provisioning service (ironic)
Install and configure the Bare Metal Provisioning service (ironic)

OpenStack Team
rhos-docs@redhat.com
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 .......................................................... 4

CHAPTER 1. BARE METAL PROVISIONING SERVICE (IRONIC) FUNCTIONALITY .......... 5

CHAPTER 2. REQUIREMENTS FOR BARE METAL PROVISIONING .......................... 7

  2.1. HARDWARE REQUIREMENTS .................................................................... 7
  2.2. NETWORKING REQUIREMENTS ............................................................... 7
    2.2.1. The default bare metal network ......................................................... 8
    2.2.2. The custom composable bare metal network ...................................... 8

CHAPTER 3. DEPLOYING AN OVERCLOUD WITH THE BARE METAL PROVISIONING SERVICE .................... 10

  3.1. CONFIGURING THE DEFAULT FLAT NETWORK ........................................ 10
  3.2. CONFIGURING A CUSTOM IPV4 PROVISIONING NETWORK ...................... 11
  3.3. CONFIGURING A CUSTOM IPV6 PROVISIONING NETWORK ..................... 12
  3.4. CONFIGURING THE OVERCLOUD TO ENABLE BARE METAL PROVISIONING .... 14
  3.5. TESTING THE BARE METAL PROVISIONING SERVICE ............................. 16
  3.6. ADDITIONAL RESOURCES ...................................................................... 17

CHAPTER 4. CONFIGURING THE BARE METAL PROVISIONING SERVICE AFTER DEPLOYMENT ............... 18

  4.1. CONFIGURING OPENSTACK NETWORKING ............................................ 18
    4.1.1. Configuring OpenStack Networking to communicate with the Bare Metal Provisioning service on a flat bare metal network .......... 18
    4.1.2. Configuring OpenStack Networking to communicate with the Bare Metal Provisioning service on a custom composable bare metal network .... 20
  4.2. CONFIGURING NODE CLEANING ........................................................... 20
    4.2.1. Cleaning nodes manually ................................................................... 21
  4.3. CREATING THE BARE METAL FLAVOR AND RESOURCE CLASS .................... 22
  4.4. CREATING THE BARE METAL IMAGES ................................................... 23
    4.4.1. Preparing the deploy images ............................................................. 23
    4.4.2. Preparing the user image ................................................................. 24
      4.4.2.1. Disk image environment variables ............................................. 24
      4.4.3. Installing the user image ............................................................... 25
  4.5. CONFIGURING DEPLOY INTERFACES ..................................................... 26
    Prerequisites ................................................................................................. 27
    Workflow ...................................................................................................... 27
    4.5.1. Configuring the direct deploy interface on the overcloud ................. 28
  4.6. ADDING PHYSICAL MACHINES AS BARE METAL NODES ......................... 29
    4.6.1. Enrolling a bare metal node with an inventory file ............................. 30
    4.6.2. Enrolling a bare-metal node manually ............................................. 32
  4.7. CONFIGURING REDFISH VIRTUAL MEDIA BOOT .................................... 35
    4.7.1. Deploying a bare metal server with Redfish virtual media boot ............ 35
  4.8. USING HOST AGGREGATES TO SEPARATE PHYSICAL AND VIRTUAL MACHINE PROVISIONING ....... 37

CHAPTER 5. ADMINISTERING BARE METAL NODES ..................................................................... 39

  5.1. LAUNCHING BARE METAL INSTANCES ................................................. 39
    5.1.1. Launching instances with the command line interface ....................... 39
    5.1.2. Launching instances with the dashboard ............................................ 40
  5.2. CONFIGURING PORT GROUPS IN THE BARE METAL PROVISIONING SERVICE ............. 40
    5.2.1. Configuring port groups on switches manually .................................. 41
    5.2.2. Configuring port groups in the Bare Metal Provisioning service ........ 41
  5.3. DETERMINING THE HOST TO IP ADDRESS MAPPING .............................. 42
  5.4. ATTACHING AND DETACHING VIRTUAL NETWORK INTERFACES .................... 44
  5.5. CONFIGURING NOTIFICATIONS FOR THE BARE METAL PROVISIONING SERVICE ............. 47
5.6. CONFIGURING AUTOMATIC POWER FAULT RECOVERY
5.7. INTROSPECTING OVERCLOUD NODES

CHAPTER 6. BOOTING FROM CINDER VOLUMES .................................................. 50
6.1. CINDER VOLUME BOOT FOR BARE METAL NODES 50
6.2. CONFIGURING NODES FOR CINDER VOLUME BOOT 50
6.3. CONFIGURING ISCSI KERNEL PARAMETERS ON THE BOOT DISK 50
6.4. CREATING AND USING A BOOT VOLUME IN CINDER 54

CHAPTER 7. ML2 NETWORKING-ANSIBLE ............................................................ 55
7.1. MODULAR LAYER 2 (ML2) NETWORKING-ANSIBLE 55
7.2. NETWORKING REQUIREMENTS FOR NETWORKING-ANSIBLE 55
7.3. OPENSTACK BARE METAL (IRONIC) REQUIREMENTS FOR NETWORKING-ANSIBLE 56
7.4. ENABLING NETWORKING-ANSIBLE ML2 FUNCTIONALITY 56
7.5. CONFIGURING NETWORKS FOR NETWORKING-ANSIBLE 58
  7.5.1. Configuring networks for networking-ansible in access mode 59
  7.5.2. Configuring ports for bare metal guests in access mode 59
  7.5.3. Configuring networks for networking-ansible in trunk mode 60
  7.5.4. Configuring ports for bare metal guests in trunk mode 61
7.6. TESTING NETWORKING-ANSIBLE ML2 FUNCTIONS 61

CHAPTER 8. TROUBLESHOOTING THE BARE METAL PROVISIONING SERVICE ............... 63
8.1. PXE BOOT ERRORS 63
8.2. LOGIN ERRORS AFTER THE BARE METAL NODE BOOTS 64
8.3. BOOT-TO-DISK ERRORS ON DEPLOYED NODES 65
8.4. THE BARE METAL PROVISIONING SERVICE DOES NOT RECEIVE THE CORRECT HOST NAME 66
8.5. INVALID OPENSTACK IDENTITY SERVICE CREDENTIALS WHEN EXECUTING BARE METAL PROVISIONING SERVICE COMMANDS 66
8.6. HARDWARE ENROLMENT 66
8.7. TROUBLESHOOTING IDRAC ISSUES 66
8.8. CONFIGURING THE SERVER CONSOLE 67

CHAPTER 9. BARE METAL DRIVERS ................................................................. 69
9.1. INTELLIGENT PLATFORM MANAGEMENT INTERFACE (IPMI) 69
9.2. REDFISH 69
9.3. DELL REMOTE ACCESS CONTROLLER (DRAC) 69
9.4. INTEGRATED REMOTE MANAGEMENT CONTROLLER (IRMC) 70
9.5. INTEGRATED LIGHTS-OUT (ILO) 70
9.6. CONVERTING TO NEXT GENERATION POWER MANAGEMENT DRIVERS 71
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.
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

![Default bare metal network architecture diagram](image)

### 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:

   ```bash
   [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:

   ```yaml
   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:

```yaml
| 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:

```yaml
| 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:

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

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

   ```bash
   (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:

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

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

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

   ```bash
   (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:

   ```yaml
   # 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: ['<ipv6_start_address>', '<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:

   ```bash
   $ 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:

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

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

   ```yaml
   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 an environment file to deploy the overcloud with the Bare Metal Provisioning service (ironic) enabled. You can use one of the example templates located in the /usr/share/openstack-tripleo-heat-templates/environments/services directory:

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

The following procedure uses the ironic-overcloud.yaml file to illustrate how to configure your deployment to enable the Bare Metal Provisioning service.

Procedure

1. Copy the example bare metal template to your environment file directory:
2. To configure a hybrid deployment where cloud users can create both bare metal and virtual instances, add `AggregateInstanceExtraSpecsFilter` to the `NovaSchedulerDefaultFilters` parameter in `ironic-overcloud.yaml`, if not already present:

```yaml
parameter_defaults:
  NovaSchedulerDefaultFilters:
    AggregateInstanceExtraSpecsFilter,ComputeFilter,ComputeCapabilitiesFilter,ImagePropertiesFilter
```

**NOTE**

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

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

```yaml
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.

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

```yaml
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.

5. To enable bare metal introspection, copy the example `ironic-inspector.yaml` file to your environment file directory:

```bash
$ cp /usr/share/openstack-tripleo-heat-templates/environments/services/ironic-inspector.yaml /home/stack/templates/ironic-inspector.yaml
```

6. Configure `ironic-inspector.yaml` for your environment:

```yaml
parameter_defaults:
  IronicInspectorSubnets:
  - ip_range: <ip_range>
```
IronicInspectorInterface: '<baremetal_interface>'

- Replace `<ip_range>` with the IP ranges for your environment. You can specify multiple ranges.

- 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`.

7. 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 /home/stack/templates/network_environment_overrides.yaml \ 
    -n /home/stack/templates/network_data.yaml \ 
    -e /home/stack/templates/ironic-overcloud.yaml \
```

**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.

### 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](https://docs.openstack.org/openstack-manuals/).

**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:

```bash
$ openstack catalog list
```

### 3.6. ADDITIONAL RESOURCES

- Deployment command options 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

After you deploy an overcloud with the Bare Metal Provisioning service (ironic), you might need to complete some additional configuration to prepare your environment for your bare metal workloads:

- Configure networking.
- Configure node cleaning.
- Create bare metal flavors and images for your bare metal nodes.
- Configure the deploy interface.
- Configure virtual media boot.
- Separate virtual and physical machine provisioning.

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.

4.1. CONFIGURING OPENSTACK NETWORKING

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

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

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

4.1.1. Configuring OpenStack Networking to communicate with the Bare Metal Provisioning service on a flat bare metal network

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

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 Identity as the administrative user:

   ```
   $ source ~/.overcloudrc
   ```
2. Create the flat network over which to provision bare metal instances:

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

Replace `NETWORK_NAME` with a name for this network. The name of the physical network over which you implement the virtual network (in this case `baremetal`) was set earlier in the `~/templates/network-environment.yaml` file, with the parameter `NeutronBridgeMappings`.

3. Create the subnet on the flat network:

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

Replace the following values:

- Replace `SUBNET_NAME` with a name for the subnet.
- Replace `NETWORK_NAME` with the name of the provisioning network 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 starting with `START_IP` and ending 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 will be allocated.
- Replace `END_IP` with the IP address that denotes the end of the range of IP addresses within the new subnet from which floating IP addresses will be allocated.

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

```
$ openstack router create ROUTER_NAME
```

Replace `ROUTER_NAME` with a name for the router.

5. Attach the subnet to the new router:

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

4.1.2. Configuring OpenStack Networking to communicate with the Bare Metal Provisioning service on a custom composable bare metal network

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

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

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

If the name of the overcloud network is not provisioning, set the IronicProvisioningNetwork parameter to provisioning and redeploy the overcloud:

```
~/templates/ironic.yaml
```

```
parameter_defaults:
  IronicProvisioningNetwork:
    default: provisioning
    description: Name or UUID of the overcloud network used for provisioning bare metal nodes, if IronicDefaultNetworkInterface is set to "neutron". The default value can be left during the initial deployment and should be changed to an actual UUID in a post-deployment stack update.
    type: string
```

4.2. CONFIGURING NODE CLEANING

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

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. To configure node cleaning, provide the provider network UUID on the Controller that hosts the Bare Metal Provisioning service:

   ~/templates/ironic.yaml

   parameter_defaults:
     IronicCleaningNetwork: <UUID>

   Replace `<UUID>` with the UUID of the bare metal network that you create in the previous steps.

   You can find the UUID with the `openstack network show` command:

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

   **NOTE**

   You must perform this configuration after the initial overcloud deployment, because the UUID for the network is not available beforehand.

2. To apply the changes, redeploy the overcloud with the `openstack overcloud deploy`. For more information about the deployment command, see Deploying an overcloud with the Bare Metal Provisioning service.

4.2.1. Cleaning nodes manually

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

Node cleaning has two modes:

- **Metadata only clean** - Removes partitions from all disks on a given node. This is a faster clean cycle, but less secure 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. This can take several hours to complete.

**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**

To initiate a **metadata** clean:

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

To initiate a **full** clean:

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

Replace `UUID` with the UUID of the node that you want to clean.
After a successful cleaning, the node state returns to manageable. If the state is clean failed, inspect the last_error field for the cause of failure.

4.3. CREATING THE BARE METAL FLAVOR AND RESOURCE CLASS

You must create a flavor and a resource class to use to tag the bare metal nodes for a particular workload.

Procedure

1. Source the overcloud credentials file:

```
$ source ~/overcloudrc
```

2. Create a new instance flavor for bare metal nodes:

```
(overcloud)$ openstack flavor create --id auto \n   --ram <ram_size_mb> --disk <disk_size_gb> \n   --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 \n   --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 new instance flavor for bare metal nodes with the custom resource class:

```
(overcloud)$ openstack flavor set \n   --property resources:CUSTOM_BAREMETAL_<CUSTOM>=1 \n   baremetal
```
To determine the name of a custom resource class that corresponds to a resource class of a Bare Metal service 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:

```bash
(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:

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

### 4.4. CREATING THE BARE METAL IMAGES

An overcloud that includes the Bare Metal Provisioning service (ironic) requires two sets of images. During deployment, the Bare Metal Provisioning service boots bare metal nodes from the deploy image, and copies the user image onto nodes.

#### The deploy image

The Bare Metal Provisioning service uses the deploy image to boot the bare metal node and copy a user image onto the bare metal node. The deploy image consists of the **kernel** image and the **ramdisk** image.

#### The user image

The user image is the image that you deploy onto the bare metal node. The user image also has a **kernel** image and **ramdisk** image, but additionally, the user image contains a **main** image. The main image is either a root partition, or a whole-disk image.

- A **whole-disk image** is an image that contains the partition table and boot loader. The Bare Metal Provisioning service does not control the subsequent reboot of a node deployed with a whole-disk image as the node supports localboot.

- A **root partition image** contains only the root partition of the operating system. If you use a root partition, after the deploy image is loaded into the Image service, you can set the deploy image as the node boot image in the node properties. A subsequent reboot of the node uses netboot to pull down the user image.

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

#### 4.4.1. Preparing the deploy images

You do not have to create the deploy image because it was already created when the overcloud was deployed by the undercloud. The deploy image consists of two images - the kernel image and the ramdisk image:
These images are often in the home directory, unless you have deleted them, or unpacked them elsewhere. If they are not in the home directory, and you still have the `rhosp-director-images-ipa` package installed, these images are in the `/usr/share/rhosp-director-images/ironic-python-agent*.tar` 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**

Extract the images and upload them to the Image service:

```bash
$ openstack image create
  --container-format aki
  --disk-format aki
  --public
  --file ./tftpboot/agent.kernel bm-deploy-kernel
$ openstack image create
  --container-format ari
  --disk-format ari
  --public
  --file ./tftpboot/agent.ramdisk bm-deploy-ramdisk
```

### 4.4.2. Preparing the user image

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

#### 4.4.2.1. Disk image environment variables

As a part of the disk image building process, the director requires a base image and registration details to obtain packages for the new overcloud image. Define these attributes with the following Linux environment variables.

**NOTE**

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

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

**DIB_LOCAL_IMAGE**

Sets the local image that you want to use as the basis for your whole disk image.

**REG_ACTIVATION_KEY**

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

**REG_AUTO.Attach**
Defines whether to attach the most compatible subscription automatically.

**REG_BASE_URL**

The base URL of the content delivery server that contains packages for the image. The default Customer Portal Subscription Management process uses `https://cdn.redhat.com`. If you use a Red Hat Satellite 6 server, set this parameter to the base URL of your Satellite server.

**REG_ENVIRONMENT**

Registers to an environment within an organization.

**REG_METHOD**

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

**REG_ORG**

The organization where you want to register the images.

**REG_POOL_ID**

The pool ID of the product subscription information.

**REG_PASSWORD**

Sets the password for the user account that registers the image.

**REG_RELEASE**

Sets the Red Hat Enterprise Linux minor release version. You must use it with the `REG_AUTO_ATTACH` or the `REG_POOL_ID` environment variable.

**REG_REPOS**

A comma-separated string of repository names. Each repository in this string is enabled through `subscription-manager`.

**REG_SAT_URL**

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

**REG_SERVER_URL**

Sets the host name of the subscription service to use. The default host name is for the Red Hat Customer Portal at `subscription.rhn.redhat.com`. If you use a Red Hat Satellite 6 server, set this parameter to the host name of your Satellite server.

**REG_USER**

Sets the user name for the account that registers the image.

### 4.4.3. Installing the user image

Configure the user image and then upload the image to the Image service (glance).

**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. Download the Red Hat Enterprise Linux KVM guest image from the Customer Portal.

2. Define `DIB_LOCAL_IMAGE` as the downloaded image:
$ export DIB_LOCAL_IMAGE=rhel-8.0-x86_64-kvm.qcow2

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

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

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

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

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

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

4. Create the user images with the `diskimage-builder` tool:

$ export DIB_RELEASE=8
$ disk-image-create rhel baremetal -o rhel-image

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

5. Upload the images to the Image service:

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

4.5. 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-vmlinux
  - 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.

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

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

   ```yaml
   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:

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

3. Include the custom environment with your overcloud deployment:

   ```bash
   $ 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:

   ```bash
   $ 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

There are two methods to enroll a bare metal node:

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

After you enroll the physical machines, Compute is not immediately notified of new resources, because the Compute resource tracker synchronizes periodically. You can view changes after the next periodic task runs. You can update the frequency of the periodic task with the `scheduler_driver_task_period`, in the `/etc/nova/nova.conf` file. The default period is 60 seconds.

### 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.

```yaml
nodes:
  - name: node0
driver: ipmi
driver_info:
ipmi_address: <ipmi_ip>
ipmi_username: <user>
ipmi_password: <password>
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.
- 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.

2. Source the `overcloudrc` file:

   ```bash
   $ source ~/overcloudrc
   ```

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

   ```bash
   $ 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:

   ```bash
   $ 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, `/var/lib/ironic/httpboot/agent.kernel`.
   - Replace `<initramfs_file>` with the path to the `.initramfs` image, for example, `/var/lib/ironic/httpboot/agent.ramdisk`.

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

   ```bash
   $ 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`:

   ```bash
   $ 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:

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

8. Check that the nodes are enrolled:
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 `IronicEnabledDrivers` 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, `/var/lib/ironic/httpboot/agent.kernel`. 

   There might be a delay between enrolling a node and its state being shown.
7. Update the node properties to match the hardware specifications on the node:

```bash
$ 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:

```bash
$ 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: If you have multiple disks, set the root device hints to inform the deploy ramdisk which disk to use for deployment:

```bash
$ 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.

10. 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:

```bash
$ 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.

11. Validate the configuration of the node:

```bash
$ openstack baremetal node validate <node>
```

<table>
<thead>
<tr>
<th>Interface</th>
<th>Result</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>boot</td>
<td>False</td>
<td>Cannot validate image information for node: a02178db-1550-4244-a2b7-d7035c743a9b because one or more parameters are missing from its instance_info. Missing: ['ramdisk', 'kernel', 'image_source']</td>
</tr>
<tr>
<td>console</td>
<td>None</td>
<td>not supported</td>
</tr>
<tr>
<td>deploy</td>
<td>False</td>
<td>Cannot validate image information for node: a02178db-1550-4244-a2b7-d7035c743a9b because one or more parameters are missing from its instance_info. Missing: ['ramdisk', 'kernel', 'image_source']</td>
</tr>
<tr>
<td>inspect</td>
<td>None</td>
<td>not supported</td>
</tr>
<tr>
<td>management</td>
<td>True</td>
<td></td>
</tr>
<tr>
<td>network</td>
<td>True</td>
<td></td>
</tr>
<tr>
<td>power</td>
<td>True</td>
<td></td>
</tr>
<tr>
<td>raid</td>
<td>True</td>
<td></td>
</tr>
<tr>
<td>storage</td>
<td>True</td>
<td></td>
</tr>
</tbody>
</table>

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.

### 4.7. 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.
Procedure

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

   ```bash
   $ 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.
   ```bash
   $ openstack flavor set baremetal --property baremetal=true
   ```
2. Set the **baremetal** property to **false** on the flavors that virtual instances use:
   ```bash
   $ openstack flavor set FLAVOR_NAME --property baremetal=false
   ```
3. Create a host aggregate called **baremetal-hosts**:
   ```bash
   $ openstack aggregate create --property baremetal=true baremetal-hosts
   ```
4. Add each Controller node to the **baremetal-hosts** aggregate:
   ```bash
   $ 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**:
   ```bash
   $ openstack aggregate create --property baremetal=false virtual-hosts
   ```
6. Add each Compute node to the **virtual-hosts** aggregate:
   ```bash
   $ 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:

```bash
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

Use the openstack command line 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. Configure the shell to access the Identity service (keystone) as the administrative user:

   $ source ~/overcloudrc

2. Deploy the instance:

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

Replace the following values:

- Replace NETWORK_UUID with the unique identifier for the network that you created to use with the Bare Metal Provisioning service.

- Replace IMAGE_UUID with the unique identifier for the disk image that was uploaded to the Image service.

- Replace INSTANCE_NAME with a name for the bare metal instance.

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

3. Check the status of the instance:

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

### 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
   ```
<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>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).</td>
</tr>
<tr>
<td>output_key</td>
<td>HostsEntry</td>
</tr>
</tbody>
</table>
| output_value | 172.17.0.10 overcloud-controller-0.localdomain overcloud-controller-0 overcloud-10.8.145.18 overcloud-controller-0.external.localdomain overcloud-controller-0.external 10.8.145.18 overcloud-controller-0.external.localdomain overcloud-controller-0.external overcloud-controller-0.172.17.0.10 controller-0.172.0.10 overcloud-controller-0.internalapi.localdomain overcloud-controller-0.internalapi 172.17.0.10 overcloud-controller-0.internalapi localdomain overcloud-controller-0.internalapi overcloud-controller-0.172.18.0.15 storage.0.localdomain overcloud-controller-0.storage 172.18.0.15 overcloud-controller-0.storage.localdomain overcloud-controller-0.storage overcloud-controller-0.172.21.2.12 overcloud-controller-0.storagemgmt.localdomain overcloud-controller-0.storagemgmt 172.21.2.12 overcloud-controller-0.storagemgmt.localdomain overcloud-controller-0.storagemgmt overcloud-controller-0.172.17.0.21 controller-0.172.17.0.21 overcloud-controller-0.internalapi.localdomain overcloud-controller-0.internalapi 172.17.0.21 overcloud-controller-0.internalapi localdomain overcloud-controller-0.internalapi overcloud-controller-0.10.8.146.13 controller-0.10.8.146.13 overcloud-controller-0.management.localdomain overcloud-controller-0.management 10.8.146.13 overcloud-controller-0.management.localdomain overcloud-controller-0.management overcloud-controller-0.10.8.146.13 controller-0.10.8.146.13 overcloud-controller-0.ctlplane.localdomain overcloud-controller-0.ctlplane 10.8.146.13 overcloud-controller-0.ctlplane.localdomain overcloud-controller-0.ctlplane overcloud-controller-0.172.17.0.21 compute-0.172.17.0.21 overcloud-compute-0.localdomain overcloud-compute-0 172.17.0.21 overcloud-compute-0.localdomain overcloud-compute-0 overcloud-compute-0.10.8.146.12 compute-0.10.8.146.12 overcloud-compute-0.external.localdomain overcloud-compute-0.external 10.8.146.12 overcloud-compute-0.external.localdomain overcloud-compute-0.external overcloud-compute-0.172.17.0.21 compute-0.172.17.0.21 overcloud-compute-0.internalapi.localdomain overcloud-compute-0.internalapi 172.17.0.21 overcloud-compute-0.internalapi localdomain overcloud-compute-0.internalapi overcloud-compute-0.172.18.0.20 compute-0.172.18.0.20 overcloud-compute-0.storage.localdomain overcloud-compute-0.storage 172.18.0.20 overcloud-compute-0.storage.localdomain overcloud-compute-0.storage overcloud-compute-0.10.8.146.12 compute-0.10.8.146.12 overcloud-compute-0.storagemgmt.localdomain overcloud-compute-0.storagemgmt 10.8.146.12 overcloud-compute-0.storagemgmt.localdomain overcloud-compute-0.storagemgmt overcloud-compute-0.172.16.0.16 compute-0.172.16.0.16 overcloud-compute-0.tenant.localdomain overcloud-compute-0.tenant 172.16.0.16 overcloud-compute-0.tenant.localdomain overcloud-compute-0.tenant overcloud-compute-0.10.8.146.12 compute-0.10.8.146.12 overcloud-compute-0.ctlplane.localdomain overcloud-compute-0.ctlplane 10.8.146.12 overcloud-compute-0.ctlplane.localdomain overcloud-compute-0.ctlplane overcloud-compute-0.10.8.145.16 localdomain 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 ~]s openstack stack output show overcloud HostsEntry -c
To map the hosts to bare metal nodes, run the following command:

```bash
(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:

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

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

```bash
$ openstack server list
```

```
+--------------------------------------+-------------------------+--------+------------------------+----------
| ID                                   | Name                    | Status | Networks               | Image     |
|--------------------------------------|-------------------------|--------|------------------------+----------|
| 53095a64-1646-4dd1-bbf3-b51cbcc38789 | overcloud-controller-2 | ACTIVE | ctlplane=192.168.24.7  | overcloud-ctlplane=192.168.24.7 |
| 3a1bc89c-5d0d-44c7-a569-f2a3b4c73d65 | overcloud-controller-0 | ACTIVE | ctlplane=192.168.24.8  | overcloud-ctlplane=192.168.24.8 |
| 6b01531a-f55d-40e9-b3a2-6d02be0b915b | overcloud-controller-1 | ACTIVE | ctlplane=192.168.24.16 | overcloud-ctlplane=192.168.24.16 |
| c61cc52b-cc48-4903-891a-073c60f53091 | overcloud-novacompute-0 | ACTIVE | ctlplane=192.168.24.24 | overcloud-ctlplane=192.168.24.24 |
```

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

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

```
+--------------------------------------+
| ID                                   |
+--------------------------------------+
| 6181c089-7e33-4f1c-b8fe-2523ff431ff6 |
```

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

```bash
$ openstack port show 6181c089-7e33-4f1c-b8fe-2523ff431ff6 -c mac_address -c fixed_ips
```

```
+-------------+------------------------------------------------------------------------------+
| Field       | Value                                                                        |
|-------------+------------------------------------------------------------------------------|
| fixed_ips   | ip_address='192.168.24.24', subnet_id='1a11c677-5946-4733-87c3-23a9e0677aa' |
| mac_address | 00:2d:28:2f:8d:95                                                             |
```

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

```bash
$ 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_versionedNotifications` 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

<table>
<thead>
<tr>
<th>Fault</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>power failure</td>
<td>The node is in maintenance mode due to power sync failures that exceed the maximum number of retries.</td>
</tr>
<tr>
<td>clean failure</td>
<td>The node is in maintenance mode due to the failure of a cleaning operation.</td>
</tr>
<tr>
<td>rescue abort failure</td>
<td>The node is in maintenance mode due to the failure of a cleaning operation during rescue abort.</td>
</tr>
<tr>
<td>none</td>
<td>There is no fault present.</td>
</tr>
</tbody>
</table>
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**

- 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:

   ```bash
   $ source ~/overcloudrc
   ```

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

   ```bash
   $ 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:

   ```bash
   $ 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
```

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:

```
```

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:

  ```yaml
  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:
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 Chapter 4, 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:

```bash
$ 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 [Chapter 4, 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:

     ```bash
     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:

     ```bash
     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
```

<table>
<thead>
<tr>
<th>Value in the above example kernel line</th>
<th>Corresponding information</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://192.168.200.2:8088">http://192.168.200.2:8088</a></td>
<td>Parameter <strong>http_url</strong> in /etc/ironic/ironic.conf file. This IP address must be on the Bare Metal network.</td>
</tr>
<tr>
<td>5a6cdbe3-2c90-4a90-b3c6-85b449b30512</td>
<td>UUID of the baremetal node in <strong>ironic node-list</strong>.</td>
</tr>
<tr>
<td>deploy_kernel</td>
<td>This is the deploy kernel image in the Image service that is copied down as /httpboot/&lt;NODE_UUID&gt;/deploy_kernel.</td>
</tr>
<tr>
<td><a href="http://192.168.200.2:6385">http://192.168.200.2:6385</a></td>
<td>Parameter <strong>api_url</strong> in /etc/ironic/ironic.conf file. This IP address must be on the Bare Metal network.</td>
</tr>
<tr>
<td>ipmi</td>
<td>The IPMI Driver in use by the Bare Metal Provisioning service for this node.</td>
</tr>
<tr>
<td>deploy_ramdisk</td>
<td>This is the deploy ramdisk image in the Image service that is copied down as /httpboot/&lt;NODE_UUID&gt;/deploy_ramdisk.</td>
</tr>
</tbody>
</table>

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`:

```yaml
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:

   ```bash
   $ 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`:

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

4. Import the image into director:

   ```bash
   $ openstack overcloud image upload --image-path overcloud-full.qcow2
   ```

5. Deploy the overcloud.

### Verification

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

   ```bash
   $ 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:

   ```bash
   [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)

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. Enable the `ipmi` driver, and set the following information in the `driver_info` of the node:

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

9.2. REDFISH

A standard RESTful API for IT infrastructure developed by the Distributed Management Task Force (DMTF)

- `redfish_username` - The Redfish username.
- `redfish_password` - The Redfish password.
- `redfish_address` - The IP address of the Redfish controller.
- `redfish_system_id` - The canonical path to the system resource. This path must include the root service, version, and the path/unique ID for the system. For example: `/redfish/v1/Systems/CX34R87`.
- `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`.

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 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.

### 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:
   ```bash
   $ 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:
   ```bash
   # dnf install python-scciclient
   ```
3. Restart the Bare Metal conductor service:
   ```bash
   # 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.

You must also install the `python-proliantutils` package and restart the Bare Metal conductor service:
# dnf install python-proliantutils
# systemctl restart openstack-ironic-conductor.service

## 9.6. CONVERTING TO NEXT GENERATION POWER MANAGEMENT DRIVERS

Red Hat OpenStack Platform now uses next generation drivers, also known as *hardware types*, that replace older drivers.

The following table shows an analogous comparison between older drivers and their next generation hardware type equivalent:

<table>
<thead>
<tr>
<th>Old driver</th>
<th>New hardware type</th>
</tr>
</thead>
<tbody>
<tr>
<td>pxe_ipmitool</td>
<td>ipmi</td>
</tr>
<tr>
<td>pxe_drac</td>
<td>idrac</td>
</tr>
<tr>
<td>pxe_ilo</td>
<td>ilo</td>
</tr>
<tr>
<td>pxe_irmc</td>
<td>irmc</td>
</tr>
<tr>
<td>fake_pxe</td>
<td>fake-hardware</td>
</tr>
</tbody>
</table>

In Red Hat OpenStack Platform (RHOSP) 15, these older drivers have been removed and are no longer accessible. You must change to the new hardware types before you upgrade to RHOSP 15.

### Procedure

1. Check the current list of enabled hardware types:

   ```
   $ source ~/overcloud
   $ openstack baremetal driver list --type dynamic
   ```

2. If you use a hardware type driver that is not enabled, use the `IronicEnabledHardwareTypes` parameter in an environment file to enable the driver:

   ```
   parameter_defaults:
   IronicEnabledHardwareTypes: ipmi,redfish,idrac
   ```

3. Save the file and run the overcloud deployment command:

   ```
   $ openstack overcloud deploy -e [ENVIRONMENT_FILE] -r [ROLES_DATA] -n [NETWORK_DATA]
   ```

   Ensure that you include all environment and data files relevant to your overcloud.

4. Run the following commands. Substitute the `OLDDRIVER` and `NEWDRIVER` variables for your power management type:
$ source ~/overcloud
$ OLDDRIVER="pxe_ipmitool"
$ NEWDRIVER="ipmi"
$ for NODE in $(openstack baremetal node list --driver $OLDDRIVER -c UUID -f value) ; do
  openstack baremetal node set $NODE --driver $NEWDRIVER; done