Red Hat OpenStack Platform 17.1

Installing and managing Red Hat OpenStack Platform with director

Using director to create and manage a Red Hat OpenStack Platform cloud

Red Hat OpenStack Platform 17.1 Installing and managing Red Hat OpenStack Platform with director

Using director to create and manage a Red Hat OpenStack Platform cloud

OpenStack Team
rhos-docs@redhat.com

Legal Notice

Abstract

Install Red Hat OpenStack Platform 17 in an enterprise environment using the Red Hat OpenStack Platform director. This includes installing the director, planning your environment, and creating an OpenStack environment with the director.

Table of Contents

MAKING OPEN SOURCE MORE INCLUSIVE

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

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

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

Providing documentation feedback in Jira

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

1. Ensure that you are logged in to Jira. If you do not have a Jira account, create an account to submit feedback.

2. Click the following link to open a the **Create Issue** page: Create Issue

3. Complete the **Summary** and **Description** fields. In the **Description** field, include the documentation URL, chapter or section number, and a detailed description of the issue. Do not modify any other fields in the form.

4. Click **Create**.

CHAPTER 1. INTRODUCTION TO DIRECTOR

Red Hat OpenStack Platform (RHOSP) director is a toolset for installing and managing a complete RHOSP environment. Director is based primarily on the OpenStack project TripleO. With director you can install a fully-operational, lean, and robust RHOSP environment that can provision and control bare metal systems to use as RHOSP nodes.

Director uses two main concepts: an undercloud and an overcloud. First you install the undercloud, and then you use the undercloud as a tool to install and configure the overcloud.
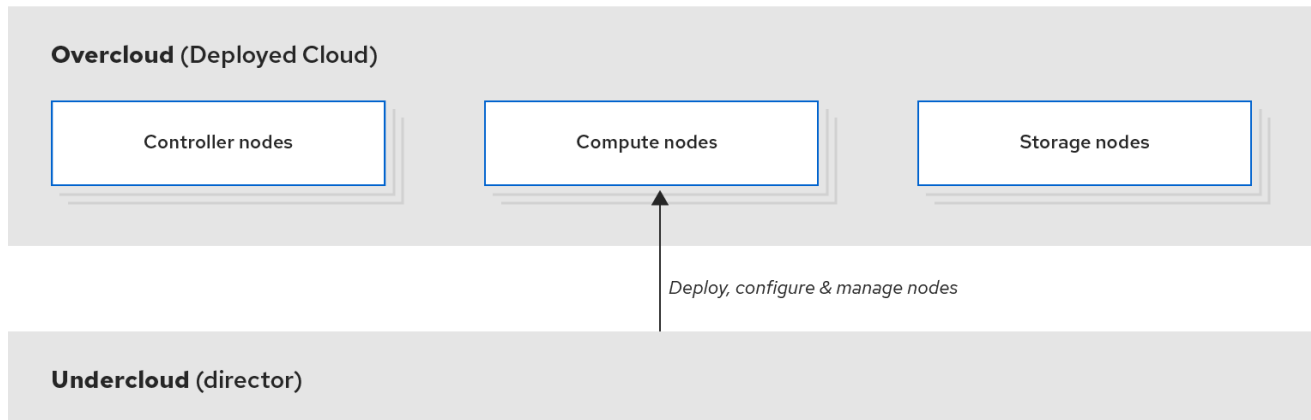
1.1. UNDERSTANDING THE UNDERCLOUD

The undercloud is the main management node that contains the Red Hat OpenStack Platform (RHOSP) director toolset. It is a single-system RHOSP installation that includes components for provisioning and managing the RHOSP nodes that form your RHOSP environment: the overcloud. The components that form the undercloud have multiple functions:

RHOSP services

> The undercloud uses RHOSP service components as its base tool set. Each service operates within a separate container on the undercloud:

> - Identity service (keystone): Provides authentication and authorization for director services.
>
> - Bare Metal Provisioning service (ironic) and Compute service (nova): Manages bare-metal nodes.
>
> - Networking service (neutron) and Open vSwitch: Control networking for bare-metal nodes.
>
> - Orchestration service (heat): Provides the orchestration of nodes after director writes the overcloud image to disk.

Environment planning

> The undercloud includes planning functions that you can use to create and assign certain node roles. The undercloud includes a default set of node roles that you can assign to specific nodes: Compute, Controller, and various Storage roles. You can also design custom roles. Additionally, you can select which RHOSP services to include on each node role, which provides a method to model new node types or isolate certain components on their own host.

Bare metal system control

The undercloud uses the out-of-band management interface, usually Intelligent Platform Management Interface (IPMI), of each node for power management control and a PXE-based service to discover hardware attributes and install RHOSP on each node. You can use this feature to provision bare metal systems as RHOSP nodes. For a full list of power management drivers, see Power management drivers.

Orchestration

The undercloud contains a set of YAML templates that represent a set of plans for your environment. The undercloud imports these plans and follows their instructions to create the resulting RHOSP environment. The plans also include hooks that you can use to incorporate your own customizations as certain points in the environment creation process.

1.2. UNDERSTANDING THE OVERCLOUD

The overcloud is the resulting Red Hat OpenStack Platform (RHOSP) environment that the undercloud creates. The overcloud consists of multiple nodes with different roles that you define based on the RHOSP environment that you want to create. The undercloud includes a default set of overcloud node roles:

Controller

Controller nodes provide administration, networking, and high availability for the RHOSP environment. A recommended RHOSP environment contains three Controller nodes together in a high availability cluster.
A default Controller node role supports the following components. Not all of these services are enabled by default. Some of these components require custom or pre-packaged environment files to enable:

- Dashboard service (horizon)

- Identity service (keystone)

- Compute service (nova)

- Networking service (neutron)

- Image Service (glance)

- Block Storage service (cinder)

- Object Storage service (swift)

- Orchestration service (heat)

- Shared File Systems service (manila)

- Bare Metal Provisioning service (ironic)

- Load Balancing-as-a-Service (octavia)

- Key Manager service (barbican)

- MariaDB

- Open vSwitch

- Pacemaker and Galera for high availability services.

Compute

Compute nodes provide computing resources for the RHOSP environment. You can add more Compute nodes to scale out your environment over time. A default Compute node contains the following components:

- Compute service (nova)

- KVM/QEMU

- Open vSwitch

Storage

Storage nodes provide storage for the RHOSP environment. The following list contains information about the various types of Storage node in RHOSP:

- Ceph Storage nodes – Used to form storage clusters. Each node contains a Ceph Object Storage Daemon (OSD). Additionally, director installs Ceph Monitor onto the Controller nodes in situations where you deploy Ceph Storage nodes as part of your environment.

- Block Storage (cinder) – Used as external block storage for highly available Controller nodes. This node contains the following components:

  - Block Storage (cinder) volume

  - Telemetry agents

  - Open vSwitch.

- Object Storage (swift) – These nodes provide an external storage layer for RHOSP Object Storage. The Controller nodes access object storage nodes through the Swift proxy. Object storage nodes contain the following components:

  - Object Storage (swift) storage

  - Telemetry agents

  - Open vSwitch.

1.3. WORKING WITH RED HAT CEPH STORAGE IN RHOSP

It is common for large organizations that use Red Hat OpenStack Platform (RHOSP) to serve thousands of clients or more. Each OpenStack client is likely to have their own unique needs when consuming block storage resources. Deploying the Image service (glance), the Block Storage service (cinder), and the Compute service (nova) on a single node can become impossible to manage in large deployments with thousands of clients. Scaling RHOSP externally resolves this challenge.

However, there is also a practical requirement to virtualize the storage layer with a solution like Red Hat Ceph Storage so that you can scale the RHOSP storage layer from tens of terabytes to petabytes, or even exabytes of storage. Red Hat Ceph Storage provides this storage virtualization layer with high availability and high performance while running on commodity hardware. While virtualization might seem like it comes with a performance penalty, Red Hat Ceph Storage stripes block device images as objects across the cluster, meaning that large Ceph Block Device images have better performance than a standalone disk. Ceph Block devices also support caching, copy-on-write cloning, and copy-on-read cloning for enhanced performance.

For more information about Red Hat Ceph Storage, see Red Hat Ceph Storage .

CHAPTER 2. PLANNING YOUR UNDERCLOUD

Before you configure and install director on the undercloud, you must plan your undercloud host to ensure it meets certain requirements.

2.1. PREPARING YOUR UNDERCLOUD NETWORKING

The undercloud requires a minimum of 2 x 1 Gbps Network Interface Cards (NICs), one for each of the following main networks:

- **Provisioning or Control Plane network** The network that director uses to provision your nodes and access them over SSH when executing Ansible configuration. This network also enables SSH access from the undercloud to overcloud nodes. The undercloud contains DHCP services for introspection and provisioning other nodes on this network, which means that no other DHCP services should exist on this network. Director configures the interface for this network.

- **External network**: Enables access to Red Hat OpenStack Platform (RHOSP) repositories, container image sources, and other servers such as DNS servers or NTP servers. Use this network for standard access to the undercloud from your workstation. You must manually configure an interface on the undercloud to access the external network.

When you plan your network, review the following guidelines:

- Red Hat recommends using one network for provisioning and the control plane and another network for the data plane.

- The provisioning and control plane network can be configured on top of a Linux bond or on individual interfaces. If you use a Linux bond, configure it as an active-backup bond type.

  - On non-controller nodes, the amount of traffic is relatively low on provisioning and control plane networks, and they do not require high bandwidth or load balancing.

  - On Controllers, the provisioning and control plane networks need additional bandwidth. The reason for increased bandwidth is that Controllers serve many nodes in other roles. More bandwidth is also required when frequent changes are made to the environment. Controllers that manage more than 50 Compute nodes, or that provision more than four bare-metal nodes simultaneously, should have 4-10 times the bandwidth of the interfaces on the non-controller nodes.

- The undercloud should have a higher bandwidth connection to the provisioning network when more than 50 overcloud nodes are provisioned.

- Do not use the same Provisioning or Control Plane NIC as the one that you use to access the director machine from your workstation. The director installation creates a bridge by using the Provisioning NIC, which drops any remote connections. Use the External NIC for remote connections to the director system.

- The Provisioning network requires an IP range that fits your environment size. Use the following guidelines to determine the total number of IP addresses to include in this range:

  - Include at least one temporary IP address for each node that connects to the Provisioning network during introspection.

  - Include at least one permanent IP address for each node that connects to the Provisioning network during deployment.

- Include an extra IP address for the virtual IP of the overcloud high availability cluster on the Provisioning network.

- Include additional IP addresses within this range for scaling the environment.

- To prevent a Controller node network card or network switch failure disrupting overcloud services availability, ensure that the keystone admin endpoint is located on a network that uses bonded network cards or networking hardware redundancy. If you move the keystone endpoint to a different network, such as **internal_api**, ensure that the undercloud can reach the VLAN or subnet. For more information, see the Red Hat Knowledgebase solution How to migrate Keystone Admin Endpoint to internal_api network.

2.2. DETERMINING ENVIRONMENT SCALE

Before you install the undercloud, determine the scale of your environment. Include the following factors when you plan your environment:

How many nodes do you want to deploy in your overcloud?

The undercloud manages each node within an overcloud. Provisioning overcloud nodes consumes resources on the undercloud. You must provide your undercloud with enough resources to adequately provision and control all of your overcloud nodes.

How many simultaneous operations do you want the undercloud to perform?

Most Red Hat OpenStack Platform (RHOSP) services on the undercloud use a set of workers. Each worker performs an operation specific to that service. Multiple workers provide simultaneous operations. The default number of workers on the undercloud is determined by halving the total CPU thread count on the undercloud. In this instance, thread count refers to the number of CPU cores multiplied by the hyper-threading value. For example, if your undercloud has a CPU with 16 threads, then the director services spawn 8 workers by default. Director also uses a set of minimum and maximum caps by default:

Service	Minimum	Maximum
Orchestration service (heat)	4	24
All other services	2	12

The undercloud has the following minimum CPU and memory requirements:

- An 8-thread 64-bit x86 processor with support for the Intel 64 or AMD64 CPU extensions. This provides 4 workers for each undercloud service.

- A minimum of 24 GB of RAM.

To use a larger number of workers, increase the vCPUs and memory of your undercloud using the following recommendations:

- **Minimum:** Use 1.5 GB of memory for each thread. For example, a machine with 48 threads requires 72 GB of RAM to provide the minimum coverage for 24 heat workers and 12 workers for other services.

- **Recommended:** Use 3 GB of memory for each thread. For example, a machine with 48 threads requires 144 GB of RAM to provide the recommended coverage for 24 heat workers and 12 workers for other services.

2.3. UNDERCLOUD DISK SIZING

The recommended minimum undercloud disk size is **100 GB** of available disk space on the root disk:

- 20 GB for container images

- 10 GB to accommodate QCOW2 image conversion and caching during the node provisioning process

- 70 GB+ for general usage, logging, metrics, and growth

2.4. VIRTUALIZED UNDERCLOUD NODE SUPPORT

Red Hat only supports a virtualized undercloud on the following platforms:

Platform	Notes
Kernel-based Virtual Machine (KVM)	Hosted by Red Hat Enterprise Linux, as listed on Certified Guest Operating Systems in Red Hat OpenStack Platform, OpenShift Virtualization and Red Hat Enterprise Linux with KVM
Microsoft Hyper-V	Hosted by versions of Hyper-V as listed on the Red Hat Customer Portal Certification Catalogue.
VMware ESX and ESXi	Hosted by versions of ESX and ESXi as listed on the Red Hat Customer Portal Certification Catalogue

> **IMPORTANT**
>
> Ensure your hypervisor supports Red Hat Enterprise Linux 9.2 guests.

Virtual machine requirements

Resource requirements for a virtual undercloud are similar to those of a bare-metal undercloud. Consider the various tuning options when provisioning such as network model, guest CPU capabilities, storage backend, storage format, and caching mode.

Network considerations

Power management

The undercloud virtual machine (VM) requires access to the overcloud nodes' power management devices. This is the IP address set for the **pm_addr** parameter when registering nodes.

Provisioning network

The NIC used for the provisioning network, **ctlplane**, requires the ability to broadcast and serve DHCP requests to the NICs of the overcloud's bare-metal nodes. Create a bridge that connects the VM's NIC to the same network as the bare metal NICs.

Allow traffic from an unknown address

You must configure your virtual undercloud hypervisor, VMware ESX or ESXi, to prevent the hypervisor blocking the undercloud from transmitting traffic from an unknown address:

- On IPv4 **ctlplane** network: Allow forged transmits.

- On IPv6 **ctlplane** network: Allow forged transmits, MAC address changes, and promiscuous mode operation.
  For more information about how to configure VMware ESX or ESXi, see Securing vSphere Standard Switches on the VMware docs website.

You must power off and on the director VM after you apply these settings. It is not sufficient to only reboot the VM.

2.5. UNDERCLOUD REPOSITORIES

You run Red Hat OpenStack Platform (RHOSP) 17.1 on Red Hat Enterprise Linux (RHEL) 9.2.

NOTE

If you synchronize repositories with Red Hat Satellite, you can enable specific versions of the Red Hat Enterprise Linux repositories. However, the repository remains the same despite the version you choose. For example, you can enable the 9.2 version of the BaseOS repository, but the repository name is still **rhel-9-for-x86_64-baseos-eus-rpms** despite the specific version you choose.

WARNING

Any repositories except the ones specified here are not supported. Unless recommended, do not enable any other products or repositories except the ones listed in the following tables or else you might encounter package dependency issues. Do not enable Extra Packages for Enterprise Linux (EPEL).

Core repositories

The following table lists core repositories for installing the undercloud.

Name	Repository	Description of requirement
Red Hat Enterprise Linux 9 for x86_64 – BaseOS (RPMs) Extended Update Support (EUS)	**rhel-9-for-x86_64-baseos-eus-rpms**	Base operating system repository for x86_64 systems.
Red Hat Enterprise Linux 9 for x86_64 – AppStream (RPMs)	**rhel-9-for-x86_64-appstream-eus-rpms**	Contains Red Hat OpenStack Platform dependencies.
Red Hat Enterprise Linux 9 for x86_64 – High Availability (RPMs) Extended Update Support (EUS)	**rhel-9-for-x86_64-highavailability-eus-rpms**	High availability tools for Red Hat Enterprise Linux. Used for Controller node high availability.

Name	Repository	Description of requirement
Red Hat OpenStack Platform for RHEL 9 (RPMs)	**openstack-17.1-for-rhel-9-x86_64-rpms**	Core Red Hat OpenStack Platform repository, which contains packages for Red Hat OpenStack Platform director.
Red Hat Fast Datapath for RHEL 9 (RPMS)	**fast-datapath-for-rhel-9-x86_64-rpms**	Provides Open vSwitch (OVS) packages for OpenStack Platform.

CHAPTER 3. PREPARING FOR DIRECTOR INSTALLATION

To install and configure director, you must complete some preparation tasks to ensure you have registered the undercloud to the Red Hat Customer Portal or a Red Hat Satellite server, you have installed the director packages, and you have configured a container image source for the director to pull container images during installation.

3.1. PREPARING THE UNDERCLOUD

Before you can install director, you must complete some basic configuration on the host machine.

Procedure

1. Log in to your undercloud as the **root** user.

2. Create the **stack** user:

   ```
   [root@director ~]# useradd stack
   ```

3. Set a password for the user:

   ```
   [root@director ~]# passwd stack
   ```

4. Disable password requirements when using **sudo**:

   ```
   [root@director ~]# echo "stack ALL=(root) NOPASSWD:ALL" | tee -a /etc/sudoers.d/stack
   [root@director ~]# chmod 0440 /etc/sudoers.d/stack
   ```

5. Switch to the new **stack** user:

   ```
   [root@director ~]# su - stack
   [stack@director ~]$
   ```

6. Create directories for system images and heat templates:

   ```
   [stack@director ~]$ mkdir ~/images
   [stack@director ~]$ mkdir ~/templates
   ```

   Director uses system images and heat templates to create the overcloud environment. Red Hat recommends creating these directories to help you organize your local file system.

7. Check the base and full hostname of the undercloud:

   ```
   [stack@director ~]$ hostname
   [stack@director ~]$ hostname -f
   ```

   If either of the previous commands do not report the correct fully-qualified hostname or report an error, use **hostnamectl** to set a hostname:

   ```
   [stack@director ~]$ sudo hostnamectl set-hostname undercloud.example.com
   ```

8. If you are not using a DNS server that can resolve the fully qualified domain name (FQDN) of

the undercloud host, edit the **/etc/hosts** and include an entry for the system hostname. The IP address in **/etc/hosts** must match the address that you plan to use for your undercloud public API. For example, if the system uses **undercloud.example.com** as the FQDN and uses **10.0.0.1** for its IP address, add the following line to the /etc/hosts file:

```
10.0.0.1  undercloud.example.com undercloud
```

9. If you plan for the Red Hat OpenStack Platform director to be on a separate domain than the overcloud or its identity provider, then you must add the additional domains to /etc/resolv.conf:

```
search overcloud.com idp.overcloud.com
```

IMPORTANT

You must enable the DNS domain for ports extension (**dns_domain_ports**) for DNS to internally resolve names for ports in your RHOSP environment. Using the **NeutronDnsDomain** default value, **openstacklocal**, means that the Networking service does not internally resolve port names for DNS. For more information, see Specifying the name that DNS assigns to ports in *Configuring Red Hat OpenStack Platform networking*.

3.2. REGISTERING THE UNDERCLOUD AND ATTACHING SUBSCRIPTIONS

Before you can install director, you must run **subscription-manager** to register the undercloud and attach a valid Red Hat OpenStack Platform subscription.

Procedure

1. Log in to your undercloud as the **stack** user.

2. Register your system either with the Red Hat Content Delivery Network or with a Red Hat Satellite. For example, run the following command to register the system to the Content Delivery Network. Enter your Customer Portal user name and password when prompted:

```
[stack@director ~]$ sudo subscription-manager register
```

3. Find the entitlement pool ID for Red Hat OpenStack Platform (RHOSP) director:

```
[stack@director ~]$ sudo subscription-manager list --available --all --matches="Red Hat OpenStack"
Subscription Name:   Name of SKU
Provides:           Red Hat Single Sign-On
                    Red Hat Enterprise Linux Workstation
                    Red Hat CloudForms
                    Red Hat OpenStack
                    Red Hat Software Collections (for RHEL Workstation)
SKU:                SKU-Number
Contract:           Contract-Number
Pool ID:            Valid-Pool-Number-123456
Provides Management: Yes
Available:          1
Suggested:           1
```

```
Service Level:      Support-level
Service Type:       Service-Type
Subscription Type:  Sub-type
Ends:               End-date
System Type:        Physical
```

4. Locate the **Pool ID** value and attach the Red Hat OpenStack Platform 17.1 entitlement:

```
[stack@director ~]$ sudo subscription-manager attach --pool=Valid-Pool-Number-123456
```

5. Lock the undercloud to Red Hat Enterprise Linux 9.2:

```
$ sudo subscription-manager release --set=9.2
```

3.3. ENABLING REPOSITORIES FOR THE UNDERCLOUD

Enable the repositories that are required for the undercloud, and update the system packages to the latest versions.

Procedure

1. Log in to your undercloud as the **stack** user.

2. Disable all default repositories, and enable the required Red Hat Enterprise Linux (RHEL) repositories:

```
[stack@director ~]$ sudo subscription-manager repos --disable=*
[stack@director ~]$ sudo subscription-manager repos \
--enable=rhel-9-for-x86_64-baseos-eus-rpms          \
--enable=rhel-9-for-x86_64-appstream-eus-rpms       \
--enable=rhel-9-for-x86_64-highavailability-eus-rpms \
--enable=openstack-17.1-for-rhel-9-x86_64-rpms \
--enable=fast-datapath-for-rhel-9-x86_64-rpms
```

These repositories contain packages that the director installation requires.

3. Perform an update on your system to ensure that you have the latest base system packages:

```
[stack@director ~]$ sudo dnf update -y
[stack@director ~]$ sudo reboot
```

4. Install the command line tools for director installation and configuration:

```
[stack@director ~]$ sudo dnf install -y python3-tripleoclient
```

3.4. PREPARING CONTAINER IMAGES

The undercloud installation requires an environment file to determine where to obtain container images and how to store them. Generate and customize the environment file that you can use to prepare your container images.

NOTE

If you need to configure specific container image versions for your undercloud, you must pin the images to a specific version. For more information, see Pinning container images for the undercloud.

Procedure

1. Log in to the undercloud host as the **stack** user.

2. Generate the default container image preparation file:

   ```
   $ openstack tripleo container image prepare default \
     --local-push-destination \
     --output-env-file containers-prepare-parameter.yaml
   ```

   This command includes the following additional options:

   - **--local-push-destination** sets the registry on the undercloud as the location for container images. This means that director pulls the necessary images from the Red Hat Container Catalog and pushes them to the registry on the undercloud. Director uses this registry as the container image source. To pull directly from the Red Hat Container Catalog, omit this option.

   - **--output-env-file** is an environment file name. The contents of this file include the parameters for preparing your container images. In this case, the name of the file is **containers-prepare-parameter.yaml**.

     NOTE

     You can use the same **containers-prepare-parameter.yaml** file to define a container image source for both the undercloud and the overcloud.

3. Modify the **containers-prepare-parameter.yaml** to suit your requirements. For more information about container image parameters, see Container image preparation parameters.

3.5. OBTAINING CONTAINER IMAGES FROM PRIVATE REGISTRIES

The **registry.redhat.io** registry requires authentication to access and pull images. To authenticate with **registry.redhat.io** and other private registries, include the **ContainerImageRegistryCredentials** and **ContainerImageRegistryLogin** parameters in your **containers-prepare-parameter.yaml** file.

ContainerImageRegistryCredentials

Some container image registries require authentication to access images. In this situation, use the **ContainerImageRegistryCredentials** parameter in your **containers-prepare-parameter.yaml** environment file. The **ContainerImageRegistryCredentials** parameter uses a set of keys based on the private registry URL. Each private registry URL uses its own key and value pair to define the username (key) and password (value). This provides a method to specify credentials for multiple private registries.

```
parameter_defaults:
  ContainerImagePrepare:
  - push_destination: true
    set:
      namespace: registry.redhat.io/...
```

```
  ...
  ContainerImageRegistryCredentials:
    registry.redhat.io:
      my_username: my_password
```

In the example, replace **my_username** and **my_password** with your authentication credentials. Instead of using your individual user credentials, Red Hat recommends creating a registry service account and using those credentials to access **registry.redhat.io** content.

To specify authentication details for multiple registries, set multiple key-pair values for each registry in **ContainerImageRegistryCredentials**:

```
parameter_defaults:
  ContainerImagePrepare:
  - push_destination: true
    set:
      namespace: registry.redhat.io/...

      ...
  - push_destination: true
    set:
      namespace: registry.internalsite.com/...

      ...
  ...
  ContainerImageRegistryCredentials:
    registry.redhat.io:
      myuser: 'p@55w0rd!'
    registry.internalsite.com:
      myuser2: '0th3rp@55w0rd!'
    '192.0.2.1:8787':
      myuser3: '@n0th3rp@55w0rd!'
```

> **IMPORTANT**
>
> The default **ContainerImagePrepare** parameter pulls container images from **registry.redhat.io**, which requires authentication.

For more information, see Red Hat Container Registry Authentication .

ContainerImageRegistryLogin

The **ContainerImageRegistryLogin** parameter is used to control whether an overcloud node system needs to log in to the remote registry to fetch the container images. This situation occurs when you want the overcloud nodes to pull images directly, rather than use the undercloud to host images.

You must set **ContainerImageRegistryLogin** to **true** if **push_destination** is set to false or not used for a given strategy.

```
parameter_defaults:
  ContainerImagePrepare:
  - push_destination: false
    set:
      namespace: registry.redhat.io/...

      ...
  ...
  ContainerImageRegistryCredentials:
```

```
    registry.redhat.io:
      myuser: 'p@55w0rd!'
  ContainerImageRegistryLogin: true
```

However, if the overcloud nodes do not have network connectivity to the registry hosts defined in **ContainerImageRegistryCredentials** and you set **ContainerImageRegistryLogin** to **true**, the deployment might fail when trying to perform a login. If the overcloud nodes do not have network connectivity to the registry hosts defined in the **ContainerImageRegistryCredentials**, set **push_destination** to **true** and **ContainerImageRegistryLogin** to **false** so that the overcloud nodes pull images from the undercloud.

```
  parameter_defaults:
    ContainerImagePrepare:
    - push_destination: true
      set:
        namespace: registry.redhat.io/...

        ...
    ...
    ContainerImageRegistryCredentials:
      registry.redhat.io:
        myuser: 'p@55w0rd!'
    ContainerImageRegistryLogin: false
```

CHAPTER 4. INSTALLING DIRECTOR ON THE UNDERCLOUD

To configure and install director, set the appropriate parameters in the **undercloud.conf** file and run the undercloud installation command. After you have installed director, import the overcloud images that director will use to write to bare metal nodes during node provisioning.

4.1. CONFIGURING THE UNDERCLOUD

You must configure the undercloud before installing director. You configure the undercloud in the **undercloud.conf** file, which director reads from the home directory of the **stack** user.

Procedure

1. Log in to the undercloud as the **stack** user.

2. Copy the sample **undercloud.conf** file to the home directory of the **stack** user:

   ```
   [stack@director ~]$ cp \
     /usr/share/python-tripleoclient/undercloud.conf.sample \
     ~/undercloud.conf
   ```

3. Modify the values of the parameters in the **undercloud.conf** file for your deployment. For information about the parameters you can use to configure the undercloud, see Undercloud configuration parameters.

   > **NOTE**
   >
   > If you omit or comment out a parameter, director uses the default value.

4. Save your **undercloud.conf** file.

4.2. UNDERCLOUD CONFIGURATION PARAMETERS

The following list contains information about parameters for configuring the **undercloud.conf** file. Keep all parameters within their relevant sections to avoid errors.

> **IMPORTANT**
>
> At minimum, you must set the **container_images_file** parameter to the environment file that contains your container image configuration. Without this parameter properly set to the appropriate file, director cannot obtain your container image rule set from the **ContainerImagePrepare** parameter nor your container registry authentication details from the **ContainerImageRegistryCredentials** parameter.

Defaults

The following parameters are defined in the **[DEFAULT]** section of the **undercloud.conf** file:

additional_architectures

A list of additional (kernel) architectures that an overcloud supports. Currently the overcloud supports only the **x86_64** architecture.

certificate_generation_ca

The **certmonger** nickname of the CA that signs the requested certificate. Use this option only if you have set the **generate_service_certificate** parameter. If you select the **local** CA, certmonger extracts the local CA certificate to **/etc/pki/ca-trust/source/anchors/cm-local-ca.pem** and adds the certificate to the trust chain.

clean_nodes

Defines whether to wipe the hard drive between deployments and after introspection.

cleanup

Delete temporary files. Set this to **False** to retain the temporary files used during deployment. The temporary files can help you debug the deployment if errors occur.

container_cli

The CLI tool for container management. Leave this parameter set to **podman**. Red Hat Enterprise Linux 9.2 only supports **podman**.

container_healthcheck_disabled

Disables containerized service health checks. Red Hat recommends that you enable health checks and leave this option set to **false**.

container_images_file

Heat environment file with container image information. This file can contain the following entries:

- Parameters for all required container images

- The **ContainerImagePrepare** parameter to drive the required image preparation. Usually the file that contains this parameter is named **containers-prepare-parameter.yaml**.

container_insecure_registries

A list of insecure registries for **podman** to use. Use this parameter if you want to pull images from another source, such as a private container registry. In most cases, **podman** has the certificates to pull container images from either the Red Hat Container Catalog or from your Satellite Server if the undercloud is registered to Satellite.

container_registry_mirror

An optional **registry-mirror** configured that **podman** uses.

custom_env_files

Additional environment files that you want to add to the undercloud installation.

deployment_user

The user who installs the undercloud. Leave this parameter unset to use the current default user **stack**.

discovery_default_driver

Sets the default driver for automatically enrolled nodes. Requires the **enable_node_discovery** parameter to be enabled and you must include the driver in the **enabled_hardware_types** list.

enable_ironic; enable_ironic_inspector; enable_tempest; enable_validations

Defines the core services that you want to enable for director. Leave these parameters set to **true**.

enable_node_discovery

Automatically enroll any unknown node that PXE-boots the introspection ramdisk. New nodes use the **fake** driver as a default but you can set **discovery_default_driver** to override. You can also use introspection rules to specify driver information for newly enrolled nodes.

enable_routed_networks

Defines whether to enable support for routed control plane networks.

enabled_hardware_types

A list of hardware types that you want to enable for the undercloud.

generate_service_certificate

Defines whether to generate an SSL/TLS certificate during the undercloud installation, which is used for the **undercloud_service_certificate** parameter. The undercloud installation saves the resulting certificate **/etc/pki/tls/certs/undercloud-[undercloud_public_vip].pem**. The CA defined in the **certificate_generation_ca** parameter signs this certificate.

heat_container_image

URL for the heat container image to use. Leave unset.

heat_native

Run host-based undercloud configuration using **heat-all**. Leave as **true**.

hieradata_override

Path to **hieradata** override file that configures Puppet hieradata on the director, providing custom configuration to services beyond the **undercloud.conf** parameters. If set, the undercloud installation copies this file to the /**etc/puppet/hieradata** directory and sets it as the first file in the hierarchy. For more information about using this feature, see Configuring hieradata on the undercloud.

inspection_extras

Defines whether to enable extra hardware collection during the inspection process. This parameter requires the **python-hardware** or **python-hardware-detect** packages on the introspection image.

inspection_interface

The bridge that director uses for node introspection. This is a custom bridge that the director configuration creates. The **LOCAL_INTERFACE** attaches to this bridge. Leave this as the default **br-ctlplane**.

inspection_runbench

Runs a set of benchmarks during node introspection. Set this parameter to **true** to enable the benchmarks. This option is necessary if you intend to perform benchmark analysis when inspecting the hardware of registered nodes.

ipv6_address_mode

IPv6 address configuration mode for the undercloud provisioning network. The following list contains the possible values for this parameter:

- dhcpv6-stateless - Address configuration using router advertisement (RA) and optional information using DHCPv6.

- dhcpv6-stateful - Address configuration and optional information using DHCPv6.

ipxe_enabled

Defines whether to use iPXE or standard PXE. The default is **true**, which enables iPXE. Set this parameter to **false** to use standard PXE. For PowerPC deployments, or for hybrid PowerPC and x86 deployments, set this value to **false**.

local_interface

The chosen interface for the director Provisioning NIC. This is also the device that director uses for DHCP and PXE boot services. Change this value to your chosen device. To see which device is connected, use the **ip addr** command. For example, this is the result of an **ip addr** command:

```
2: em0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen
1000
    link/ether 52:54:00:75:24:09 brd ff:ff:ff:ff:ff:ff
    inet 192.168.122.178/24 brd 192.168.122.255 scope global dynamic em0
      valid_lft 3462sec preferred_lft 3462sec
```

```
       inet6 fe80::5054:ff:fe75:2409/64 scope link
          valid_lft forever preferred_lft forever
    3: em1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noop state DOWN
          link/ether 42:0b:c2:a5:c1:26 brd ff:ff:ff:ff:ff:ff
```

In this example, the External NIC uses **em0** and the Provisioning NIC uses **em1**, which is currently not configured. In this case, set the **local_interface** to **em1**. The configuration script attaches this interface to a custom bridge defined with the **inspection_interface** parameter.

local_ip

The IP address defined for the director Provisioning NIC. This is also the IP address that director uses for DHCP and PXE boot services. Leave this value as the default **192.168.24.1/24** unless you use a different subnet for the Provisioning network, for example, if this IP address conflicts with an existing IP address or subnet in your environment.
For IPv6, the local IP address prefix length must be **/64** to support both stateful and stateless connections.

local_mtu

The maximum transmission unit (MTU) that you want to use for the **local_interface**. Do not exceed 1500 for the undercloud.

local_subnet

The local subnet that you want to use for PXE boot and DHCP interfaces. The **local_ip** address should reside in this subnet. The default is **ctlplane-subnet**.

net_config_override

Path to network configuration override template. If you set this parameter, the undercloud uses a JSON or YAML format template to configure the networking with **os-net-config** and ignores the network parameters set in **undercloud.conf**. Use this parameter when you want to configure bonding or add an option to the interface. For more information about customizing undercloud network interfaces, see Configuring undercloud network interfaces.

networks_file

Networks file to override for **heat**.

output_dir

Directory to output state, processed heat templates, and Ansible deployment files.

overcloud_domain_name

The DNS domain name that you want to use when you deploy the overcloud.

> **NOTE**
>
> When you configure the overcloud, you must set the **CloudDomain** parameter to a matching value. Set this parameter in an environment file when you configure your overcloud.

roles_file

The roles file that you want to use to override the default roles file for undercloud installation. It is highly recommended to leave this parameter unset so that the director installation uses the default roles file.

scheduler_max_attempts

The maximum number of times that the scheduler attempts to deploy an instance. This value must be greater or equal to the number of bare metal nodes that you expect to deploy at once to avoid potential race conditions when scheduling.

service_principal

The Kerberos principal for the service using the certificate. Use this parameter only if your CA requires a Kerberos principal, such as in FreeIPA.

subnets

List of routed network subnets for provisioning and introspection. The default value includes only the **ctlplane-subnet** subnet. For more information, see Subnets.

templates

Heat templates file to override.

undercloud_admin_host

The IP address or hostname defined for director admin API endpoints over SSL/TLS. The director configuration attaches the IP address to the director software bridge as a routed IP address, which uses the **/32** netmask.
If the **undercloud_admin_host** is not in the same IP network as the **local_ip**, you must configure the interface on which you want the admin APIs on the undercloud to listen. By default, the admin APIs listen on the **br-ctlplane** interface. For information about how to configure undercloud network interfaces, see Configuring undercloud network interfaces.

undercloud_debug

Sets the log level of undercloud services to **DEBUG**. Set this value to **true** to enable **DEBUG** log level.

undercloud_enable_selinux

Enable or disable SELinux during the deployment. It is highly recommended to leave this value set to **true** unless you are debugging an issue.

undercloud_hostname

Defines the fully qualified host name for the undercloud. If set, the undercloud installation configures all system host name settings. If left unset, the undercloud uses the current host name, but you must configure all system host name settings appropriately.

undercloud_log_file

The path to a log file to store the undercloud install and upgrade logs. By default, the log file is **install-undercloud.log** in the home directory. For example, **/home/stack/install-undercloud.log**.

undercloud_nameservers

A list of DNS nameservers to use for the undercloud hostname resolution.

undercloud_ntp_servers

A list of network time protocol servers to help synchronize the undercloud date and time.

undercloud_public_host

The IP address or hostname defined for director public API endpoints over SSL/TLS. The director configuration attaches the IP address to the director software bridge as a routed IP address, which uses the **/32** netmask.
If the **undercloud_public_host** is not in the same IP network as the **local_ip**, you must set the **PublicVirtualInterface** parameter to the public-facing interface on which you want the public APIs on the undercloud to listen. By default, the public APIs listen on the **br-ctlplane** interface. Set the **PublicVirtualInterface** parameter in a custom environment file, and include the custom environment file in the **undercloud.conf** file by configuring the **custom_env_files** parameter.

For information about customizing undercloud network interfaces, see Configuring undercloud network interfaces.

undercloud_service_certificate

The location and filename of the certificate for OpenStack SSL/TLS communication. Ideally, you obtain this certificate from a trusted certificate authority. Otherwise, generate your own self-signed certificate.

undercloud_timezone

Host timezone for the undercloud. If you do not specify a timezone, director uses the existing timezone configuration.

undercloud_update_packages

Defines whether to update packages during the undercloud installation.

Subnets

Each provisioning subnet is a named section in the **undercloud.conf** file. For example, to create a subnet called **ctlplane-subnet**, use the following sample in your **undercloud.conf** file:

```
[ctlplane-subnet]
cidr = 192.168.24.0/24
dhcp_start = 192.168.24.5
dhcp_end = 192.168.24.24
inspection_iprange = 192.168.24.100,192.168.24.120
gateway = 192.168.24.1
masquerade = true
```

You can specify as many provisioning networks as necessary to suit your environment.

> **IMPORTANT**
>
> Director cannot change the IP addresses for a subnet after director creates the subnet.

cidr

The network that director uses to manage overcloud instances. This is the Provisioning network, which the undercloud **neutron** service manages. Leave this as the default **192.168.24.0/24** unless you use a different subnet for the Provisioning network.

masquerade

Defines whether to masquerade the network defined in the **cidr** for external access. This provides the Provisioning network with network address translation (NAT) so that the Provisioning network has external access through director.

> **NOTE**
>
> The director configuration also enables IP forwarding automatically using the relevant **sysctl** kernel parameter.

dhcp_start; dhcp_end

The start and end of the DHCP allocation range for overcloud nodes. Ensure that this range contains enough IP addresses to allocate to your nodes. If not specified for the subnet, director determines the allocation pools by removing the values set for the **local_ip**, **gateway**, **undercloud_admin_host**, **undercloud_public_host**, and **inspection_iprange** parameters from the subnets full IP range.

You can configure non-contiguous allocation pools for undercloud control plane subnets by specifying a list of start and end address pairs. Alternatively, you can use the **dhcp_exclude** option to exclude IP addresses within an IP address range. For example, the following configurations both create allocation pools **172.20.0.100-172.20.0.150** and **172.20.0.200-172.20.0.250**:

Option 1

```
dhcp_start = 172.20.0.100,172.20.0.200
dhcp_end = 172.20.0.150,172.20.0.250
```

Option 2

```
dhcp_start = 172.20.0.100
dhcp_end = 172.20.0.250
dhcp_exclude = 172.20.0.151-172.20.0.199
```

dhcp_exclude

IP addresses to exclude in the DHCP allocation range. For example, the following configuration excludes the IP address **172.20.0.105** and the IP address range **172.20.0.210-172.20.0.219**:

```
dhcp_exclude = 172.20.0.105,172.20.0.210-172.20.0.219
```

dns_nameservers

DNS nameservers specific to the subnet. If no nameservers are defined for the subnet, the subnet uses nameservers defined in the **undercloud_nameservers** parameter.

gateway

The gateway for the overcloud instances. This is the undercloud host, which forwards traffic to the External network. Leave this as the default **192.168.24.1** unless you use a different IP address for director or want to use an external gateway directly.

host_routes

Host routes for the Neutron-managed subnet for the overcloud instances on this network. This also configures the host routes for the **local_subnet** on the undercloud.

inspection_iprange

Temporary IP range for nodes on this network to use during the inspection process. This range must not overlap with the range defined by **dhcp_start** and **dhcp_end** but must be in the same IP subnet.

4.3. CONFIGURING THE UNDERCLOUD WITH ENVIRONMENT FILES

You configure the main parameters for the undercloud through the **undercloud.conf** file. You can also perform additional undercloud configuration with an environment file that contains heat parameters.

Procedure

1. Create an environment file named **/home/stack/templates/custom-undercloud-params.yaml**.

2. Edit this file and include your heat parameters. For example, to enable debugging for certain OpenStack Platform services include the following snippet in the **custom-undercloud-params.yaml** file:

```
parameter_defaults:
    Debug: True
```

For information on the heat parameters you can configure for the undercloud, see Common heat parameters for undercloud configuration.

3. Save the environment file.

4. Edit your **undercloud.conf** file and scroll to the **custom_env_files** parameter. Edit the parameter to point to your **custom-undercloud-params.yaml** environment file:

```
custom_env_files = /home/stack/templates/custom-undercloud-params.yaml
```

> **NOTE**
>
> You can specify multiple environment files using a comma-separated list.

The director installation includes this environment file during the next undercloud installation or upgrade operation.

4.4. COMMON HEAT PARAMETERS FOR UNDERCLOUD CONFIGURATION

The following table contains some common heat parameters that you might set in a custom environment file for your undercloud.

Parameter	Description
AdminPassword	Sets the undercloud **admin** user password.
AdminEmail	Sets the undercloud **admin** user email address.
Debug	Enables debug mode.

Set these parameters in your custom environment file under the **parameter_defaults** section:

```
parameter_defaults:
  Debug: True
  AdminPassword: "myp@ssw0rd!"
  AdminEmail: "admin@example.com"
```

4.5. CONFIGURING HIERADATA ON THE UNDERCLOUD

You can provide custom configuration for services beyond the available **undercloud.conf** parameters by configuring Puppet hieradata on the director.

Procedure

1. Create a hieradata override file, for example, **/home/stack/hieradata.yaml**.

2. Add the customized hieradata to the file. For example, add the following snippet to modify the Compute (nova) service parameter **force_raw_images** from the default value of **True** to **False**:

```
nova::compute::force_raw_images: False
```

If there is no Puppet implementation for the parameter you want to set, then use the following method to configure the parameter:

```
nova::config::nova_config:
  DEFAULT/<parameter_name>:
    value: <parameter_value>
```

For example:

```
nova::config::nova_config:
  DEFAULT/network_allocate_retries:
    value: 20
  ironic/serial_console_state_timeout:
    value: 15
```

3. Set the **hieradata_override** parameter in the **undercloud.conf** file to the path of the new **/home/stack/hieradata.yaml** file:

```
hieradata_override = /home/stack/hieradata.yaml
```

4.6. INSTALLING DIRECTOR

Complete the following steps to install director and perform some basic post-installation tasks.

Procedure

1. Run the following command to install director on the undercloud:

```
[stack@director ~]$ openstack undercloud install
```

This command launches the director configuration script. Director installs additional packages, configures its services according to the configuration in the **undercloud.conf**, and starts all the RHOSP service containers. This script takes several minutes to complete.

The script generates two files:

- **/home/stack/tripleo-deploy/undercloud/tripleo-undercloud-passwords.yaml** - A list of all passwords for the director services.

- **/home/stack/stackrc** - A set of initialization variables to help you access the director command line tools.

2. Confirm that the RHOSP service containers are running:

```
[stack@director ~]$ sudo podman ps -a --format "{{.Names}} {{.Status}}"
```

The following command output indicates that the RHOSP service containers are running (**Up**):

```
memcached Up 3 hours (healthy)
haproxy Up 3 hours
rabbitmq Up 3 hours (healthy)
mysql Up 3 hours (healthy)
iscsid Up 3 hours (healthy)
keystone Up 3 hours (healthy)
keystone_cron Up 3 hours (healthy)
neutron_api Up 3 hours (healthy)
logrotate_crond Up 3 hours (healthy)
neutron_dhcp Up 3 hours (healthy)
neutron_l3_agent Up 3 hours (healthy)
neutron_ovs_agent Up 3 hours (healthy)
ironic_api Up 3 hours (healthy)
ironic_conductor Up 3 hours (healthy)
ironic_neutron_agent Up 3 hours (healthy)
ironic_pxe_tftp Up 3 hours (healthy)
ironic_pxe_http Up 3 hours (unhealthy)
ironic_inspector Up 3 hours (healthy)
ironic_inspector_dnsmasq Up 3 hours (healthy)
neutron-dnsmasq-qdhcp-30d628e6-45e6-499d-8003-28c0bc066487 Up 3 hours
...
```

3. To initialize the **stack** user to use the command line tools, run the following command:

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

The prompt now indicates that OpenStack commands authenticate and execute against the undercloud;

```
(undercloud) [stack@director ~]$
```

The director installation is complete. You can now use the director command line tools.

4.7. OBTAINING IMAGES FOR OVERCLOUD NODES

Director requires several disk images to provision overcloud nodes:

- An introspection kernel and ramdisk for bare metal system introspection over PXE boot.

- A deployment kernel and ramdisk for system provisioning and deployment.

- An overcloud kernel, ramdisk, and full image, which form a base overcloud system that director writes to the hard disk of the node.

You can obtain and install the images you need. You can also obtain and install a basic image to provision a bare OS when you do not want to run any other Red Hat OpenStack Platform (RHOSP) services or consume one of your subscription entitlements.

NOTE

If your RHOSP deployment uses IPv6, you must modify the overcloud image to disable the **cloud-init** network configuration. For more information on modifying an image, see the Red Hat Knowledgebase solution Modifying the Red Hat Linux OpenStack Platform Overcloud Image with virt-customize.

4.7.1. Installing the overcloud images

Your Red Hat OpenStack Platform (RHOSP) installation includes packages that provide you with the **overcloud-hardened-uefi-full.qcow2** overcloud image for director. This image is necessary for deployment of the overcloud with the default CPU architecture, x86–64. Importing this image into director also installs introspection images on the director PXE server.

Procedure

1. Log in to the undercloud as the **stack** user.

2. Source the **stackrc** file:

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

3. Install the **rhosp-director-images-uefi-x86_64** and **rhosp-director-images-ipa-x86_64** packages:

   ```
   (undercloud) [stack@director ~]$ sudo dnf install rhosp-director-images-uefi-x86_64 rhosp-director-images-ipa-x86_64
   ```

4. Create the **images** directory in the home directory of the **stack** user, **/home/stack/images**:

   ```
   (undercloud) [stack@director ~]$ mkdir /home/stack/images
   ```

   Skip this step if the directory already exists.

5. Extract the images archives to the **images** directory:

   ```
   (undercloud) [stack@director ~]$ cd ~/images
   (undercloud) [stack@director images]$ for i in /usr/share/rhosp-director-images/ironic-python-agent-latest.tar /usr/share/rhosp-director-images/overcloud-hardened-uefi-full-latest.tar; do tar -xvf $i; done
   ```

6. Import the images into director:

   ```
   (undercloud) [stack@director images]$ openstack overcloud image upload --image-path /home/stack/images/
   ```

   This command converts the image format from QCOW to RAW, and provides verbose updates on the status of the image upload progress.

7. Verify that the overcloud images are copied to **/var/lib/ironic/images/**:

   ```
   (undercloud) [stack@director images]$ ls -l /var/lib/ironic/images/
   total 1955660
   -rw-r--r--. 1 root 42422 40442450944 Jan 29 11:59 overcloud-hardened-uefi-full.raw
   ```

8. Verify that director has copied the introspection PXE images to **/var/lib/ironic/httpboot**:

   ```
   (undercloud) [stack@director images]$ ls -l /var/lib/ironic/httpboot
   total 417296
   -rwxr-xr-x. 1 root  root    6639920 Jan 29 14:48 agent.kernel
   ```

```
-rw-r--r--. 1 root  root  420656424 Jan 29 14:48 agent.ramdisk
-rw-r--r--. 1 42422 42422      758 Jan 29 14:29 boot.ipxe
-rw-r--r--. 1 42422 42422      488 Jan 29 14:16 inspector.ipxe
```

4.7.2. Minimal overcloud image

You can use the **overcloud-minimal** image to provision a bare OS where you do not want to run any other Red Hat OpenStack Platform (RHOSP) services or consume one of your subscription entitlements.

Your RHOSP installation includes the **overcloud-minimal** package that provides you with the following overcloud images for director:

- **overcloud-minimal**

- **overcloud-minimal-initrd**

- **overcloud-minimal-vmlinuz**

Procedure

1. Log in to the undercloud as the **stack** user.

2. Source the **stackrc** file:

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

3. Install the **overcloud-minimal** package:

   ```
   (undercloud) [stack@director ~]$ sudo dnf install rhosp-director-images-minimal
   ```

4. Extract the images archives to the **images** directory in the home directory of the **stack** user
   (/**home**/**stack**/**images**):

   ```
   (undercloud) [stack@director ~]$ cd ~/images
   (undercloud) [stack@director images]$ tar xf /usr/share/rhosp-director-images/overcloud-
   minimal-latest-17.1.tar
   ```

5. Import the images into director:

   ```
   (undercloud) [stack@director images]$ openstack overcloud image upload --image-path
   /home/stack/images/ --image-type os --os-image-name overcloud-minimal.qcow2
   ```

   The command provides updates on the status of the image upload progress:

   ```
   Image "file:///var/lib/ironic/images/overcloud-minimal.vmlinuz" was copied.
   +---------------------------------------------------+------------------+----------+
   |                   Path                            |      Name        |  Size    |
   +---------------------------------------------------+------------------+----------+
   | file:///var/lib/ironic/images/overcloud-minimal.vmlinuz | overcloud-minimal | 11172880 |
   +---------------------------------------------------+------------------+----------+
   Image "file:///var/lib/ironic/images/overcloud-minimal.initrd" was copied.
   +---------------------------------------------------+------------------+----------+
   |                   Path                            |      Name        |  Size    |
   ```

```
+-----------------------------------------------------+-----------------+----------+
| file:///var/lib/ironic/images/overcloud-minimal.initrd | overcloud-minimal | 63575845 |
+-----------------------------------------------------+-----------------+----------+
Image "file:///var/lib/ironic/images/overcloud-minimal.raw" was copied.
+-----------------------------------------------------+-----------------+-----------+
|                    Path                             |      Name       |   Size    |
+-----------------------------------------------------+-----------------+-----------+
| file:///var/lib/ironic/images/overcloud-minimal.raw | overcloud-minimal | 2912878592 |
+-----------------------------------------------------+-----------------+-----------+
```

4.8. UPDATING THE UNDERCLOUD CONFIGURATION

If you need to change the undercloud configuration to suit new requirements, you can make changes to your undercloud configuration after installation, edit the relevant configuration files and re-run the **openstack undercloud install** command.

Procedure

1. Modify the undercloud configuration files. For example, edit the **undercloud.conf** file and add the **idrac** hardware type to the list of enabled hardware types:

   ```
   enabled_hardware_types = ipmi,redfish,idrac
   ```

2. Run the **openstack undercloud install** command to refresh your undercloud with the new changes:

   ```
   [stack@director ~]$ openstack undercloud install
   ```

   Wait until the command runs to completion.

3. Initialize the **stack** user to use the command line tools,:

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

   The prompt now indicates that OpenStack commands authenticate and execute against the undercloud:

   ```
   (undercloud) [stack@director ~]$
   ```

4. Verify that director has applied the new configuration. For this example, check the list of enabled hardware types:

   ```
   (undercloud) [stack@director ~]$ openstack baremetal driver list
   +--------------------+---------------------+
   | Supported driver(s) | Active host(s)     |
   +--------------------+---------------------+
   | idrac              | director.example.com |
   | ipmi               | director.example.com |
   | redfish            | director.example.com |
   +--------------------+---------------------+
   ```

The undercloud re-configuration is complete.

4.9. UNDERCLOUD CONTAINER REGISTRY

Red Hat Enterprise Linux 9.2 no longer includes the **docker-distribution** package, which installed a Docker Registry v2. To maintain the compatibility and the same level of feature, the director installation creates an Apache web server with a vhost called **image-serve** to provide a registry. This registry also uses port 8787/TCP with SSL disabled. The Apache-based registry is not containerized, which means that you must run the following command to restart the registry:

```
$ sudo systemctl restart httpd
```

You can find the container registry logs in the following locations:

- /var/log/httpd/image_serve_access.log

- /var/log/httpd/image_serve_error.log.

The image content is served from **/var/lib/image-serve**. This location uses a specific directory layout and **apache** configuration to implement the pull function of the registry REST API.

The Apache-based registry does not support **podman push** nor **buildah push** commands, which means that you cannot push container images using traditional methods. To modify images during deployment, use the container preparation workflow, such as the **ContainerImagePrepare** parameter. To manage container images, use the container management commands:

openstack tripleo container image list

Lists all images stored on the registry.

openstack tripleo container image show

Show metadata for a specific image on the registry.

openstack tripleo container image push

Push an image from a remote registry to the undercloud registry.

openstack tripleo container image delete

Delete an image from the registry.

4.10. CONTENTS OF THE DEFAULT UNDERCLOUD DIRECTORY

In RHOSP 17, you can find all the configuration files in a single directory. The name of the directory is a combination of the openstack command used and the name of the stack. The directories have default locations but you can change the default locations by using the **--working-dir** option. You can use this option with any **tripleoclient** command that reads or creates files used with the deployment.

Default location	Command
$HOME/tripleo-deploy/undercloud	**undercloud install**, which is based on**tripleo deploy**
$HOME/tripleo-deploy/<stack>	**tripleo deploy**, <stack> is **standalone** by default
$HOME/overcloud-deploy/<stack>	**overcloud deploy**, <stack> is **overcloud** by default

The following table details the files and directories contained in the ~/**tripleo-deploy**/**undercloud** directory.

Table 4.1. Undercloud directory contents

Directory	Description
heat-launcher	The ephemeral Heat working directory containing the ephemeral Heat configuration and database backups.
install-undercloud.log	The undercloud install and upgrade logs.
tripleo-ansible-inventory.yaml	Ansible inventory for the overcloud.
tripleo-config-generated-env-files	Contains the generated environment files.
tripleo-undercloud-outputs.yaml	Contains saved undercloud outputs.
tripleo-undercloud-passwords.yaml	Contains the undercloud passwords.
undercloud-install-*.tar.bzip2	A tarball of the working directory, for example, **undercloud-install-20220823210316.tar.bzip2**.

CHAPTER 5. PLANNING YOUR OVERCLOUD

The following section contains some guidelines for planning various aspects of your Red Hat OpenStack Platform (RHOSP) environment. This includes defining node roles, planning your network topology, and storage.

> **IMPORTANT**
>
> Do not rename your overcloud nodes after they have been deployed. Renaming a node after deployment creates issues with instance management.

5.1. NODE ROLES

Director includes the following default node types to build your overcloud:

Controller

Provides key services for controlling your environment. This includes the dashboard (horizon), authentication (keystone), image storage (glance), networking (neutron), orchestration (heat), and high availability services. A Red Hat OpenStack Platform (RHOSP) environment requires three Controller nodes for a highly available production-level environment.

> **NOTE**
>
> Use environments with one Controller node only for testing purposes, not for production. Environments with two Controller nodes or more than three Controller nodes are not supported.

Compute

A physical server that acts as a hypervisor and contains the processing capabilities required to run virtual machines in the environment. A basic RHOSP environment requires at least one Compute node.

Ceph Storage

A host that provides Red Hat Ceph Storage. Additional Ceph Storage hosts scale into a cluster. This deployment role is optional.

Swift Storage

A host that provides external object storage to the OpenStack Object Storage (swift) service. This deployment role is optional.

The following table contains some examples of different overclouds and defines the node types for each scenario.

Table 5.1. Node deployment roles for scenarios

	Controller	Compute	Ceph Storage	Swift Storage	Total
Small overcloud	3	1	–	–	4
Medium overcloud	3	3	–	–	6

	Controller	Compute	Ceph Storage	Swift Storage	Total
Medium overcloud with additional object storage	3	3	-	3	9
Medium overcloud with Ceph Storage cluster	3	3	3	-	9

In addition, consider whether to split individual services into custom roles. For more information about the composable roles architecture, see Composable services and custom roles in the *Customizing your Red Hat OpenStack Platform deployment* guide.

Table 5.2. Node Deployment Roles for Proof of Concept Deployment

	Undercloud	Controller	Compute	Ceph Storage	Total
Proof of concept	1	1	1	1	4

> **WARNING**
>
> The Red Hat OpenStack Platform maintains an operational Ceph Storage cluster during day-2 operations. Therefore, some day-2 operations, such as upgrades or minor updates of the Ceph Storage cluster, are not possible in deployments with fewer than three MONs or three storage nodes. If you use a single Controller node or a single Ceph Storage node, day-2 operations will fail.

5.2. OVERCLOUD NETWORKS

It is important to plan the networking topology and subnets in your environment so that you can map roles and services to communicate with each other correctly. Red Hat OpenStack Platform (RHOSP) uses the Openstack Networking (neutron) service, which operates autonomously and manages software-based networks, static and floating IP addresses, and DHCP.

By default, director configures nodes to use the **Provisioning / Control Plane** for connectivity. However, it is possible to isolate network traffic into a series of composable networks, that you can customize and assign services.

In a typical RHOSP installation, the number of network types often exceeds the number of physical network links. To connect all the networks to the proper hosts, the overcloud uses VLAN tagging to deliver more than one network on each interface. Most of the networks are isolated subnets but some

networks require a Layer 3 gateway to provide routing for Internet access or infrastructure network connectivity. If you use VLANs to isolate your network traffic types, you must use a switch that supports 802.1Q standards to provide tagged VLANs.
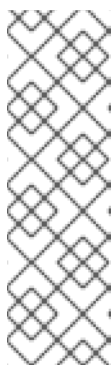
> **NOTE**
>
> You create project (tenant) networks using VLANs. You can create Geneve tunnels for special-use networks without consuming project VLANs. Red Hat recommends that you deploy a project network tunneled with Geneve, even if you intend to deploy your overcloud in neutron VLAN mode with tunneling disabled. If you deploy a project network tunneled with Geneve, you can still update your environment to use tunnel networks as utility networks or virtualization networks. It is possible to add Geneve capability to a deployment with a project VLAN, but it is not possible to add a project VLAN to an existing overcloud without causing disruption.

Director also includes a set of templates that you can use to configure NICs with isolated composable networks. The following configurations are the default configurations:

- Single NIC configuration – One NIC for the Provisioning network on the native VLAN and tagged VLANs that use subnets for the different overcloud network types.

- Bonded NIC configuration – One NIC for the Provisioning network on the native VLAN and two NICs in a bond for tagged VLANs for the different overcloud network types.

- Multiple NIC configuration – Each NIC uses a subnet for a different overcloud network type.

You can also create your own templates to map a specific NIC configuration.

The following details are also important when you consider your network configuration:

- During the overcloud creation, you refer to NICs using a single name across all overcloud machines. Ideally, you should use the same NIC on each overcloud node for each respective network to avoid confusion. For example, use the primary NIC for the Provisioning network and the secondary NIC for the OpenStack services.

- Set all overcloud systems to PXE boot off the Provisioning NIC, and disable PXE boot on the External NIC and any other NICs on the system. Also ensure that the Provisioning NIC has PXE boot at the top of the boot order, ahead of hard disks and CD/DVD drives.

- All overcloud bare metal systems require a supported power management interface, such as an Intelligent Platform Management Interface (IPMI), so that director can control the power management of each node.

- Make a note of the following details for each overcloud system: the MAC address of the Provisioning NIC, the IP address of the IPMI NIC, IPMI username, and IPMI password. This information is useful later when you configure the overcloud nodes.

- If an instance must be accessible from the external internet, you can allocate a floating IP address from a public network and associate the floating IP with an instance. The instance retains its private IP but network traffic uses NAT to traverse through to the floating IP address. Note that a floating IP address can be assigned only to a single instance rather than multiple private IP addresses. However, the floating IP address is reserved for use only by a single tenant, which means that the tenant can associate or disassociate the floating IP address with a particular instance as required. This configuration exposes your infrastructure to the external internet and you must follow suitable security practices.

- To mitigate the risk of network loops in Open vSwitch, only a single interface or a single bond can be a member of a given bridge. If you require multiple bonds or interfaces, you can configure multiple bridges.

- Red Hat recommends using DNS hostname resolution so that your overcloud nodes can connect to external services, such as the Red Hat Content Delivery Network and network time servers.

- Red Hat recommends that the Provisioning interface, External interface, and any floating IP interfaces be left at the default MTU of 1500. Connectivity problems are likely to occur otherwise. This is because routers typically cannot forward jumbo frames across Layer 3 boundaries.

5.3. OVERCLOUD STORAGE

You can use Red Hat Ceph Storage nodes as the back end storage for your overcloud environment. You can configure your overcloud to use the Ceph nodes for the following types of storage:

Images

The Image service (glance) manages the images that are used for creating virtual machine instances. Images are immutable binary blobs. You can use the Image service to store images in a Ceph Block Device. For information about supported image formats, see The Image service (glance) in *Creating and managing images*.

Volumes

The Block Storage service (cinder) manages persistent storage volumes for instances. The Block Storage service volumes are block devices. You can use a volume to boot an instance, and you can attach volumes to running instances. You can use the Block Storage service to boot a virtual machine using a copy-on-write clone of an image.

Objects

The Ceph Object Gateway (RGW) provides the default overcloud object storage on the Ceph cluster when your overcloud storage back end is Red Hat Ceph Storage. If your overcloud does not have Red Hat Ceph Storage, then the overcloud uses the Object Storage service (swift) to provide object storage. You can dedicate overcloud nodes to the Object Storage service. This is useful in situations where you need to scale or replace Controller nodes in your overcloud environment but need to retain object storage outside of a high availability cluster.

File Systems

The Shared File Systems service (manila) manages shared file systems. You can use the Shared File Systems service to manage shares backed by a CephFS file system with data on the Ceph Storage nodes.

Instance disks

When you launch an instance, the instance disk is stored as a file in the instance directory of the hypervisor. The default file location is **/var/lib/nova/instances**.

For more information about Ceph Storage, see the Red Hat Ceph Storage Architecture Guide .

5.3.1. Configuration considerations for overcloud storage

Consider the following issues when planning your storage configuration:

Instance security and performance

Using LVM on an instance that uses a back-end Block Storage volume causes issues with performance, volume visibility and availability, and data corruption. Use an LVM filter to mitigate

issues. For more information, see Enabling LVM2 filtering on overcloud nodes in *Configuring persistent storage*, and the Red Hat Knowledgebase solution Using LVM on a cinder volume exposes the data to the compute host.

Local disk partition sizes

Consider the storage and retention requirements for your deployment to determine if the following default disk partition sizes meet your requirements:

Partition	Default size
/	8GB
/tmp	1GB
/var/log	10GB
/var/log/audit	2GB
/home	1GB
/var	Node role dependent: • Object Storage nodes: 10% of remaining disk size. • Controller nodes: 90% of remaining disk size. • Non-Object Storage nodes: Allocated the remaining size of the disk after all other partitions are allocated.
/srv	On Object Storage nodes: Allocated the remaining size of the disk after all other partitions are allocated.

To change the allocated disk size for a partition, update the **role_growvols_args** extra Ansible variable in the **ansible_playbooks** definition in your **overcloud-baremetal-deploy.yaml** node definition file. For more information, see Configuring whole disk partitions for the Object Storage service.

5.4. OVERCLOUD SECURITY

Your OpenStack Platform implementation is only as secure as your environment. Follow good security principles in your networking environment to ensure that you control network access properly:

- Use network segmentation to mitigate network movement and isolate sensitive data. A flat network is much less secure.

- Restrict services access and ports to a minimum.

- Enforce proper firewall rules and password usage.

- Ensure that SELinux is enabled.

For more information about securing your system, see the following Red Hat guides:

- [Security Hardening](#) for Red Hat Enterprise Linux 9

- [Using SELinux](#) for Red Hat Enterprise Linux 9

5.5. OVERCLOUD HIGH AVAILABILITY

To deploy a highly-available overcloud, director configures multiple Controller, Compute and Storage nodes to work together as a single cluster. In case of node failure, an automated fencing and re-spawning process is triggered based on the type of node that failed. For more information about overcloud high availability architecture and services, see [Managing high availability services](#).

> **NOTE**
>
> Deploying a highly available overcloud without STONITH is not supported. You must configure a STONITH device for each node that is a part of the Pacemaker cluster in a highly available overcloud. For more information on STONITH and Pacemaker, see [Fencing in a Red Hat High Availability Cluster](#) and [Support Policies for RHEL High Availability Clusters](#).

You can also configure high availability for Compute instances with director (Instance HA). This high availability mechanism automates evacuation and re-spawning of instances on Compute nodes in case of node failure. The requirements for Instance HA are the same as the general overcloud requirements, but you must perform a few additional steps to prepare your environment for the deployment. For more information about Instance HA and installation instructions, see the [Configuring high availability for instances](#) guide.

5.6. CONTROLLER NODE REQUIREMENTS

Controller nodes host the core services in a Red Hat OpenStack Platform environment, such as the Dashboard (horizon), the back-end database server, the Identity service (keystone) authentication, and high availability services.

Processor

64-bit x86 processor with support for the Intel 64 or AMD64 CPU extensions.

Memory

The minimum amount of memory is 32 GB. However, the amount of recommended memory depends on the number of vCPUs, which is based on the number of CPU cores multiplied by hyper-threading value. Use the following calculations to determine your RAM requirements:

- **Controller RAM minimum calculation:**

  - Use 1.5 GB of memory for each vCPU. For example, a machine with 48 vCPUs should have 72 GB of RAM.

- **Controller RAM recommended calculation:**

  - Use 3 GB of memory for each vCPU. For example, a machine with 48 vCPUs should have 144 GB of RAM

For more information about measuring memory requirements, see ["Red Hat OpenStack Platform Hardware Requirements for Highly Available Controllers"](#) on the Red Hat Customer Portal.

Disk Storage and layout

A minimum amount of 50 GB storage is required if the Object Storage service (swift) is not running on the Controller nodes. However, the Telemetry and Object Storage services are both installed on the Controllers, with both configured to use the root disk. These defaults are suitable for deploying small overclouds built on commodity hardware. These environments are typical of proof-of-concept and test environments. You can use these defaults to deploy overclouds with minimal planning, but they offer little in terms of workload capacity and performance.

In an enterprise environment, however, the defaults could cause a significant bottleneck because Telemetry accesses storage constantly. This results in heavy disk I/O usage, which severely impacts the performance of all other Controller services. In this type of environment, you must plan your overcloud and configure it accordingly.

Network Interface Cards

A minimum of 2 x 1 Gbps Network Interface Cards. Use additional network interface cards for bonded interfaces or to delegate tagged VLAN traffic.

Power management

Each Controller node requires a supported power management interface, such as an Intelligent Platform Management Interface (IPMI) functionality, on the server motherboard.

5.6.1. Constraints when using NUMA

The Compute service (nova) forces a strict memory affinity for all virtual machines (VMs) that have a non-uniform memory access (NUMA) topology. This means that NUMA VMs have memory affined to the same host NUMA node as its CPUs. Do not run NUMA and non-NUMA VMs on the same hosts. If a non-NUMA VM is already running a host, and a NUMA VM affined boots on that host, this might result in out of memory (OOM) events because the NUMA VM cannot access the host memory, and is limited to its NUMA node. To avoid OOM events, ensure that NUMA-aware memory tracking is enabled on all NUMA-affined instances. To do this, configure the **hw:mem_page_size** flavor extra spec.

5.7. COMPUTE NODE REQUIREMENTS

Compute nodes are responsible for running virtual machine instances after they are launched. Compute nodes require bare metal systems that support hardware virtualization. Compute nodes must also have enough memory and disk space to support the requirements of the virtual machine instances that they host.

Processor

64-bit x86 processor with support for the Intel 64 or AMD64 CPU extensions, and the AMD-V or Intel VT hardware virtualization extensions enabled. It is recommended that this processor has a minimum of 4 cores.

Memory

A minimum of 6 GB of RAM for the host operating system, plus additional memory to accommodate for the following considerations:

- Add additional memory that you intend to make available to virtual machine instances.

- Add additional memory to run special features or additional resources on the host, such as additional kernel modules, virtual switches, monitoring solutions, and other additional background tasks.

- If you intend to use non-uniform memory access (NUMA), Red Hat recommends 8GB per CPU socket node or 16 GB per socket node if you have more then 256 GB of physical RAM.

- Configure at least 4 GB of swap space.

Disk space

A minimum of 50 GB of available disk space.

Network Interface Cards

A minimum of one 1 Gbps Network Interface Cards, although it is recommended to use at least two NICs in a production environment. Use additional network interface cards for bonded interfaces or to delegate tagged VLAN traffic.

Power management

Each Compute node requires a supported power management interface, such as an Intelligent Platform Management Interface (IPMI) functionality, on the server motherboard.

5.8. RED HAT CEPH STORAGE NODE REQUIREMENTS

There are additional node requirements using director to create a Ceph Storage cluster:

- Hardware requirements including processor, memory, and network interface card selection and disk layout are available in the Red Hat Ceph Storage Hardware Guide .

- Each Ceph Storage node requires a supported power management interface, such as Intelligent Platform Management Interface (IPMI) functionality, on the motherboard of the server.

- Each Ceph Storage node must have at least two disks. RHOSP director uses **cephadm** to deploy the Ceph Storage cluster. The cephadm functionality does not support installing Ceph OSD on the root disk of the node.

5.8.1. Red Hat Ceph Storage nodes and RHEL compatibility

RHOSP 17.1 is supported on RHEL 9.2. However, hosts that are mapped to the Red Hat Ceph Storage role update to the latest major RHEL release. Before upgrading, review the Red Hat Knowledgebase article Red Hat Ceph Storage: Supported configurations .

5.9. OBJECT STORAGE NODE REQUIREMENTS

Object Storage nodes provide an object storage layer for the overcloud. The Object Storage proxy is installed on Controller nodes. The storage layer requires bare metal nodes with multiple disks on each node.

Processor

64-bit x86 processor with support for the Intel 64 or AMD64 CPU extensions.

Memory

Memory requirements depend on the amount of storage space. Use at minimum 1 GB of memory for each 1 TB of hard disk space. For optimal performance, it is recommended to use 2 GB for each 1 TB of hard disk space, especially for workloads with files smaller than 100GB.

Disk space

Storage requirements depend on the capacity needed for the workload. It is recommended to use SSD drives to store the account and container data. The capacity ratio of account and container data to objects is approximately 1 per cent. For example, for every 100TB of hard drive capacity, provide 1TB of SSD capacity for account and container data.

However, this depends on the type of stored data. If you want to store mostly small objects, provide more SSD space. For large objects (videos, backups), use less SSD space.

Disk layout

The recommended node configuration requires a disk layout similar to the following example:

- **/dev/sda** – The root disk. Director copies the main overcloud image to the disk.

- **/dev/sdb** – Used for account data.

- **/dev/sdc** – Used for container data.

- **/dev/sdd** and onward – The object server disks. Use as many disks as necessary for your storage requirements.

Network Interface Cards

A minimum of 2 x 1 Gbps Network Interface Cards. Use additional network interface cards for bonded interfaces or to delegate tagged VLAN traffic.

Power management

Each Controller node requires a supported power management interface, such as an Intelligent Platform Management Interface (IPMI) functionality, on the server motherboard.

5.10. OVERCLOUD REPOSITORIES

You run Red Hat OpenStack Platform (RHOSP) 17.1 on Red Hat Enterprise Linux (RHEL) 9.2.

NOTE

If you synchronize repositories with Red Hat Satellite, you can enable specific versions of the Red Hat Enterprise Linux repositories. However, the repository remains the same despite the version you choose. For example, you can enable the 9.2 version of the BaseOS repository, but the repository name is still **rhel-9-for-x86_64-baseos-eus-rpms** despite the specific version you choose.

WARNING

Any repositories except the ones specified here are not supported. Unless recommended, do not enable any other products or repositories except the ones listed in the following tables or else you might encounter package dependency issues. Do not enable Extra Packages for Enterprise Linux (EPEL).

Controller node repositories

The following table lists core repositories for Controller nodes in the overcloud.

Name	Repository	Description of requirement
Red Hat Enterprise Linux 9 for x86_64 – BaseOS (RPMs) Extended Update Support (EUS)	**rhel-9-for-x86_64-baseos-eus-rpms**	Base operating system repository for x86_64 systems.

Name	Repository	Description of requirement
Red Hat Enterprise Linux 9 for x86_64 – AppStream (RPMs)	**rhel-9-for-x86_64-appstream-eus-rpms**	Contains Red Hat OpenStack Platform dependencies.
Red Hat Enterprise Linux 9 for x86_64 – High Availability (RPMs) Extended Update Support (EUS)	**rhel-9-for-x86_64-highavailability-eus-rpms**	High availability tools for Red Hat Enterprise Linux.
Red Hat OpenStack Platform for RHEL 9 x86_64 (RPMs)	**openstack-17.1-for-rhel-9-x86_64-rpms**	Core Red Hat OpenStack Platform repository.
Red Hat Fast Datapath for RHEL 9 (RPMS)	**fast-datapath-for-rhel-9-x86_64-rpms**	Provides Open vSwitch (OVS) packages for OpenStack Platform.
Red Hat Ceph Storage Tools 6 for RHEL 9 x86_64 (RPMs)	**rhceph-6-tools-for-rhel-9-x86_64-rpms**	Tools for Red Hat Ceph Storage 6 for Red Hat Enterprise Linux 9.

Compute and ComputeHCI node repositories

The following table lists core repositories for Compute and ComputeHCI nodes in the overcloud.

Name	Repository	Description of requirement
Red Hat Enterprise Linux 9 for x86_64 – BaseOS (RPMs) Extended Update Support (EUS)	**rhel-9-for-x86_64-baseos-eus-rpms**	Base operating system repository for x86_64 systems.
Red Hat Enterprise Linux 9 for x86_64 – AppStream (RPMs)	**rhel-9-for-x86_64-appstream-eus-rpms**	Contains Red Hat OpenStack Platform dependencies.
Red Hat Enterprise Linux 9 for x86_64 – High Availability (RPMs) Extended Update Support (EUS)	**rhel-9-for-x86_64-highavailability-eus-rpms**	High availability tools for Red Hat Enterprise Linux.
Red Hat OpenStack Platform for RHEL 9 x86_64 (RPMs)	**openstack-17.1-for-rhel-9-x86_64-rpms**	Core Red Hat OpenStack Platform repository.
Red Hat Fast Datapath for RHEL 9 (RPMS)	**fast-datapath-for-rhel-9-x86_64-rpms**	Provides Open vSwitch (OVS) packages for OpenStack Platform.
Red Hat Ceph Storage Tools 6 for RHEL 9 x86_64 (RPMs)	**rhceph-6-tools-for-rhel-9-x86_64-rpms**	Tools for Red Hat Ceph Storage 6 for Red Hat Enterprise Linux 9.

Ceph Storage node repositories

The following table lists Ceph Storage related repositories for the overcloud.

Name	Repository	Description of requirement
Red Hat Enterprise Linux 9 for x86_64 – BaseOS (RPMs)	**rhel-9-for-x86_64-baseos-rpms**	Base operating system repository for x86_64 systems.
Red Hat Enterprise Linux 9 for x86_64 – AppStream (RPMs)	**rhel-9-for-x86_64-appstream-rpms**	Contains Red Hat OpenStack Platform dependencies.
Red Hat OpenStack Platform Deployment Tools for RHEL 9 x86_64 (RPMs)	**openstack-17.1-deployment-tools-for-rhel-9-x86_64-rpms**	Packages to help director configure Ceph Storage nodes. This repository is included with standalone Ceph Storage subscriptions. If you use a combined OpenStack Platform and Ceph Storage subscription, use the **openstack-17.1-for-rhel-9-x86_64-rpms** repository.
Red Hat OpenStack Platform for RHEL 9 x86_64 (RPMs)	**openstack-17.1-for-rhel-9-x86_64-rpms**	Packages to help director configure Ceph Storage nodes. This repository is included with combined Red Hat OpenStack Platform and Red Hat Ceph Storage subscriptions. If you use a standalone Red Hat Ceph Storage subscription, use the **openstack-17.1-deployment-tools-for-rhel-9-x86_64-rpms** repository.
Red Hat Ceph Storage Tools 6 for RHEL 9 x86_64 (RPMs)	**rhceph-6-tools-for-rhel-9-x86_64-rpms**	Provides tools for nodes to communicate with the Ceph Storage cluster.
Red Hat Fast Datapath for RHEL 9 (RPMS)	**fast-datapath-for-rhel-9-x86_64-rpms**	Provides Open vSwitch (OVS) packages for OpenStack Platform. If you are using OVS on Ceph Storage nodes, add this repository to the network interface configuration (NIC) templates.

5.11. NODE PROVISIONING AND CONFIGURATION

You provision the overcloud nodes for your Red Hat OpenStack Platform (RHOSP) environment by using either the OpenStack Bare Metal (ironic) service, or an external tool. When your nodes are provisioned, you configure them by using director.

Provisioning with the OpenStack Bare Metal (ironic) service

Provisioning overcloud nodes by using the Bare Metal service is the standard provisioning method. For more information, see Provisioning bare metal overcloud nodes .

Provisioning with an external tool

You can use an external tool, such as Red Hat Satellite, to provision overcloud nodes. This is useful if you want to create an overcloud without power management control, use networks that have DHCP/PXE boot restrictions, or if you want to use nodes that have a custom partitioning layout that does not rely on the **overcloud-hardened-uefi-full.qcow2** image. This provisioning method does not use the OpenStack Bare Metal service (ironic) for managing nodes. For more information, see Configuring a basic overcloud with pre-provisioned nodes .

CHAPTER 6. CONFIGURING OVERCLOUD NETWORKING

To configure the physical network for your overcloud, create a network configuration file, **network_data.yaml**, that follows the structure defined in the network data schema.

6.1. DEFINING THE OVERCLOUD NETWORKS

Red Hat OpenStack Platform (RHOSP) provides default overcloud networks that you can use to host specific types of network traffic in isolation. If no isolated networks are configured, RHOSP uses the provisioning network for all services.

You can define the following isolated overcloud networks:

IPMI

Network used for power management of nodes. This network is predefined before the installation of the undercloud.

Provisioning

Director uses this network for deployment and management. The provisioning network is normally configured on a dedicated interface. The initial deployment uses DHCP with PXE, then the network is converted to static IP. By default, PXE boot must occur on the native VLAN, although some system controllers allow booting from a VLAN. By default, the compute and storage nodes use the provisioning interface as their default gateway for DNS, NTP, and system maintenance.
The undercloud can be used as a default gateway. However, all traffic is behind an IP masquerade NAT (Network Address Translation), and is not reachable from the rest of the RHOSP network. The undercloud is also a single point of failure for the overcloud default route. If there is an external gateway configured on a router device on the provisioning network, the undercloud neutron DHCP server can offer that service instead.

Internal API

The Internal API network is used for communication between the RHOSP services using API communication, RPC messages, and database communication.

Tenant

The Networking service (neutron) provides each tenant (project) with their own networks using one of the following methods:

- VLAN segregation, where each tenant network is a network VLAN.

- Tunneling through VXLAN or GRE.

Network traffic is isolated within each tenant network. Each tenant network has an IP subnet associated with it, and network namespaces means that multiple tenant networks can use the same address range without causing conflicts.

Storage

Network used for block Storage, NFS, iSCSI, and others. Ideally, this would be isolated to an entirely separate switch fabric for performance reasons.

Storage Management

The Object Storage service (swift) uses this network to synchronize data objects between participating replica nodes. The proxy service acts as the intermediary interface between user requests and the underlying storage layer. The proxy receives incoming requests and locates the necessary replica to retrieve the requested data. Services that use a Red Hat Ceph Storage back end

connect over the Storage Management network because they do not interact directly with Red Hat Ceph Storage, but instead use the front end service. The RBD driver is an exception, as this traffic connects directly to Red Hat Ceph Storage.

External

Hosts the Dashboard service (horizon) for graphical system management, the public APIs for RHOSP services, and performs SNAT for incoming traffic destined for instances.

Floating IP

Allows incoming traffic to reach instances using 1-to-1 IP address mapping between the floating IP address, and the IP address actually assigned to the instance in the tenant network. If hosting the Floating IPs on a VLAN separate from external, you can trunk the Floating IP VLAN to the Controller nodes and add the VLAN through the Networking Service (neutron) after overcloud creation. This provides a means to create multiple Floating IP networks attached to multiple bridges. The VLANs are trunked but are not configured as interfaces. Instead, the Networking Service (neutron) creates an OVS port with the VLAN segmentation ID on the chosen bridge for each Floating IP network.

> **NOTE**
>
> The provisioning network must be a native VLAN, the other networks can be trunked.

You must define specific isolated networks for each role:

Role	Network
Controller	provisioning, internal API, storage, storage management, tenant, external
Compute	provisioning, internal API, storage, tenant
Ceph Storage	provisioning, internal API, storage, storage management
Cinder Storage	provisioning, internal API, storage, storage management
Swift Storage	provisioning, internal API, storage, storage management

6.2. CREATING THE NETWORK DEFINITION FILE

Create a network definition file to configure your isolated overcloud networks.

Procedure

1. Copy the sample network definition template you require from **/usr/share/openstack-tripleo-heat-templates/network-data-samples** to your environment file directory:

   ```
   $ cp /usr/share/openstack-tripleo-heat-templates/network-data-samples/<sample_network_definition_file> /home/stack/templates/<networks_definition_file>
   ```

   - Replace **<sample_network_definition_file>** with the name of the sample network definition file that you want to copy, for example, **default-network-isolation-ipv6.yaml**.

- Replace **<networks_definition_file>** with the name of your network definition file, for example, **network_data.yaml**.

2. Update the networks and network attributes in your local network definition file to match the requirements for your overcloud network environment. For information about the properties you can use to configure network attributes in your network definition file, see Network definition file configuration options. For an example node definition file, see Example network definition file.

3. Optional: To change the user-visible names of the default networks, change the **name_lower** field to the new name for the network and update the **ServiceNetMap** with the new name:

```
- name: InternalApi
  name_lower: <custom_name>
  service_net_map_replace: <default_name>
```

- Do not modify the **name** field.

- Replace **<custom_name>** with the new name you want to assign to the network, for example, **MyCustomInternalApi**.

- Replace **<default_name>** with the default value of the **name_lower** parameter, for example, **internal_api**.

4. Optional: Add custom networks to your network definition file. The following example adds a network for storage backups:

```
- name: StorageBackup
  vip: false
  name_lower: storage_backup
  subnets:
    storage_backup_subnet:
      ip_subnet: 172.16.6.0/24
      allocation_pools:
        - start: 172.16.6.4
        - end: 172.16.6.250
      gateway_ip: 172.16.6.1
```

6.3. CREATING THE NETWORK VIP DEFINITION FILE

Create a network Virtual IP (VIP) definition file to configure the network VIPs for your overcloud.

Procedure

1. Copy the sample network VIP definition template you require from **/usr/share/openstack-tripleo-heat-templates/network-data-samples** to your environment file directory:

```
$ cp /usr/share/openstack-tripleo-heat-templates/network-data-
samples/<sample_network_VIP_definition_file>
/home/stack/templates/<network_VIP_definition_file>
```

- Replace **<sample_network_VIP_definition_file>** with the name of the sample network VIP definition file that you want to copy, for example, **vip-data-default-network-isolation.yaml**.

- Replace **<network_VIP_definition_file>** with the name of your network VIP definition file, for example, **vip_data.yaml**.

2. Optional: Configure your network VIP definition file for your environment:

```
- network: <network>
  dns_name: <dns_name>
  name: <vip_name>
  ip_address: <vip_address>
  subnet: <subnet>
```

- Replace **<network>** with the name of the network where the IP address is allocated, for example, **internal_api** or **storage**.

- Optional: Replace **<dns_name>** with the FQDN (Fully Qualified Domain Name), for example, **overcloud**.

- Optional: Replace **<vip_name>** with the VIP name.

- Optional: Replace **<vip_address>** with the virtual IP address for the network.

- Optional: Replace **<subnet>** with the name of the subnet to use when creating the virtual IP port. Required for routed networks.
  For example, the following configuration defines the external network and control plane VIPs:

```
- network: external
  dns_name: overcloud
  ip_address: '1.2.3.4'
- network: ctlplane
  dns_name: overcloud
```

For information about the properties you can use to configure your network VIPs, see Network VIP attribute properties.

6.4. NETWORK DEFINITION FILE CONFIGURATION OPTIONS

You can use the properties in the following tables to configure network attributes in your **network_data.yaml** file.

Table 6.1. Network definition properties

Property	Value
name	The name of the network.
name_lower	The lowercase version of the network name. Director maps this name to respective networks assigned to roles in the **roles_data.yaml** file. The default value is **name.lower()**.

Property	Value
allocation_pools	The IP range for the IPv4 subnet. Default value: start: 192.0.5.100 end: 192.0.5.150
ipv6_allocation_pools	The IP range for the IPv6 subnet. Default value: start: 2001:db6:fd00:1000:100::1 end: 2001:db6:fd00:1000:150::1
routes	List of IPv4 networks that require routing through the network gateway.
routes_ipv6	List of IPv6 networks that require routing through the network gateway.
vlan	The VLAN ID for the network.

NOTE

The **routes** and **routes_ipv6** options contain a list of routes. Each route is a dictionary entry with the **destination** and **nexthop** keys. Both options are of type string.

```
routes:
  - destination: 198.51.100.0/24
    nexthop: 192.0.5.1
  - destination: 203.0.113.0/24
    nexthost: 192.0.5.1
```

```
routes:
  - destination: 2001:db6:fd00:2000::/64
    nexthop: 2001:db6:fd00:1000:100::1
  - destination: 2001:db6:fd00:3000::/64
    nexthost: 2001:db6:fd00:1000:100::1
```

6.5. NETWORK VIP ATTRIBUTE PROPERTIES

You can use the following properties to configure network VIP attributes in your **network_data.yaml** file.

Table 6.3. Network VIP attribute properties

Property	Value
name	The virtual IP name. The default value is **$network_name_virtual_ip**.
network	(Mandatory) The network name.
ip_address	The fixed IP address of the VIP.

Property	Value
subnet	The subnet name. Specifies the subnet for the virtual IP port. Required for deployments that use routed networks.
dns_name	The FQDN (Fully Qualified Domain Name). The default value is **overcloud**.

6.6. EXAMPLE NETWORK DEFINITION FILE

The following example network definition file configures the storage, storage management, internal API, tenant, and external networks.

```
- name: Storage
  name_lower: storage
  vip: true
  ipv6: true
  mtu: 1500
  subnets:
    storage_subnet:
      ipv6_subnet: fd00:fd00:fd00:3000::/64
      ipv6_allocation_pools:
        - start: fd00:fd00:fd00:3000::10
          end: fd00:fd00:fd00:3000:ffff:ffff:ffff:fffe
      vlan: 30
- name: StorageMgmt
  name_lower: storage_mgmt
  vip: true
  ipv6: true
  mtu: 1500
  subnets:
    storage_mgmt_subnet:
      ipv6_subnet: fd00:fd00:fd00:4000::/64
      ipv6_allocation_pools:
        - start: fd00:fd00:fd00:4000::10
          end: fd00:fd00:fd00:4000:ffff:ffff:ffff:fffe
      vlan: 40
- name: InternalApi
  name_lower: internal_api
  vip: true
  ipv6: true
  mtu: 1500
  subnets:
    internal_api_subnet:
      ipv6_subnet: fd00:fd00:fd00:2000::/64
      ipv6_allocation_pools:
        - start: fd00:fd00:fd00:2000::10
          end: fd00:fd00:fd00:2000:ffff:ffff:ffff:fffe
      vlan: 20
- name: Tenant
  name_lower: tenant
  vip: false # Tenant networks do not use VIPs
  ipv6: true
  mtu: 1500
```

```
  subnets:
    tenant_subnet:
      ipv6_subnet: fd00:fd00:fd00:5000::/64
      ipv6_allocation_pools:
        - start: fd00:fd00:fd00:5000::10
          end: fd00:fd00:fd00:5000:ffff:ffff:ffff:fffe
      vlan: 50
- name: External
  name_lower: external
  vip: true
  ipv6: true
  mtu: 1500
  subnets:
    external_subnet:
      ipv6_subnet: 2001:db8:fd00:1000::/64
      ipv6_allocation_pools:
        - start: 2001:db8:fd00:1000::10
          end: 2001:db8:fd00:1000:ffff:ffff:ffff:fffe
      gateway_ipv6: 2001:db8:fd00:1000::1
      vlan: 10
```

CHAPTER 7. PROVISIONING AND DEPLOYING YOUR OVERCLOUD

To create an overcloud you must perform the following tasks:

1. Provision the network resources for your physical networks:

   a. If you are deploying network isolation or a custom composable network, then create a network definition file in YAML format.

   b. Run the network provisioning command, including the network definition file.

   c. Create a network Virtual IP (VIP) definition file in YAML format.

   d. Run the network VIP provisioning command, including the network VIP definition file.

2. Provision your bare metal nodes:

   a. Create a node definition file in YAML format.

   b. Run the bare-metal node provisioning command, including the node definition file.

3. Deploy your overcloud.

   a. Run the deployment command, including the heat environment files that the provisioning commands generate.

7.1. PROVISIONING THE OVERCLOUD NETWORKS

To configure the network resources for your Red Hat OpenStack Platform (RHOSP) physical network environment, you must perform the following tasks:

1. Configure and provision the network resources for your overcloud.

2. Configure and provision the network Virtual IPs for your overcloud.

7.1.1. Configuring and provisioning overcloud network definitions

You configure the physical network for your overcloud in a network definition file in YAML format. The provisioning process creates a heat environment file from your network definition file that contains your network specifications. When you deploy your overcloud, include this heat environment file in the deployment command.

Prerequisites

- The undercloud is installed. For more information, see Installing director .

Procedure

1. Source the **stackrc** undercloud credential file:

   ```
   $ source ~/stackrc
   ```

2. Copy the sample network definition template you require from **/usr/share/openstack-tripleo-heat-templates/network-data-samples** to your environment file directory:

```
(undercloud)$ cp /usr/share/openstack-tripleo-heat-templates/network-data-samples/default-
network-isolation.yaml /home/stack/templates/network_data.yaml
```

3. Configure your network definition file for your network environment. For example, you can update the external network definition:

```
- name: External
  name_lower: external
  vip: true
  mtu: 1500
  subnets:
    external_subnet:
      ip_subnet: 10.0.0.0/24
      allocation_pools:
        - start: 10.0.0.4
          end: 10.0.0.250
      gateway_ip: 10.0.0.1
      vlan: 10
```

4. Configure any other networks and network attributes for your environment. For more information about the properties you can use to configure network attributes in your network definition file, see Configuring overcloud networking .

5. Provision the overcloud networks:

```
(undercloud)$ openstack overcloud network provision \
[--templates <templates_directory> \]
--output  <deployment_file> \
/home/stack/templates/<networks_definition_file>
```

   - Optional: Include the **--templates** option to use your own templates instead of the default templates located in **/usr/share/openstack-tripleo-heat-templates**. Replace **<templates_directory>** with the path to the directory that contains your templates.

   - Replace **<deployment_file>** with the name of the heat environment file to generate for inclusion in the deployment command, for example **/home/stack/templates/overcloud-networks-deployed.yaml**.

   - Replace **<networks_definition_file>** with the name of your networks definition file, for example, **network_data.yaml**.

6. When network provisioning is complete, you can use the following commands to check the created networks and subnets:

```
(undercloud)$ openstack network list
(undercloud)$ openstack subnet list
(undercloud)$ openstack network show <network>
(undercloud)$ openstack subnet show <subnet>
```

   - Replace **<network>** with the name or UUID of the network you want to check.

   - Replace **<subnet>** with the name or UUID of the subnet you want to check.

Next steps

- Configuring and provisioning network VIPs for the overcloud

7.1.2. Configuring and provisioning network VIPs for the overcloud

You configure the network Virtual IPs (VIPs) for your overcloud in a network VIP definition file in YAML format. The provisioning process creates a heat environment file from your VIP definition file that contains your VIP specifications. When you deploy your overcloud, include this heat environment file in the deployment command.

Prerequisites

- The undercloud is installed. For more information, see Installing director.

- Your overcloud networks are provisioned. For more information, see Configuring and provisioning overcloud network definitions.

Procedure

1. Source the **stackrc** undercloud credential file:

   ```
   $ source ~/stackrc
   ```

2. Copy the sample network VIP definition template you require from **/usr/share/openstack-tripleo-heat-templates/network-data-samples** to your environment file directory:

   ```
   (undercloud)$ cp /usr/share/openstack-tripleo-heat-templates/network-data-samples/vip-data-default-network-isolation.yaml /home/stack/templates/vip_data.yaml
   ```

3. Optional: Configure your VIP definition file for your environment. For example, the following defines the external network and control plane VIPs:

   ```
   - name: external_vip
     network: external
     ip_address: 10.0.0.0
     subnet: external_vip_subnet
     dns_name: overcloud
   - name: ctlplane_vip
     network: ctlplane
     ip_address: 192.168.122.0
     subnet: ctlplane_vip_subnet
     dns_name: overcloud
   ```

   For more information about the properties you can use to configure network VIP attributes in your VIP definition file, see Network VIP attribute properties.

4. Provision the network VIPs:

   ```
   (undercloud)$ openstack overcloud network vip provision \
    [--templates <templates_directory> \]
    --stack <stack> \
    --output <deployment_file> \
    /home/stack/templates/<vip_definition_file>
   ```

- Optional: Include the **--templates** option to use your own templates instead of the default templates located in **/usr/share/openstack-tripleo-heat-templates**. Replace **<templates_directory>** with the path to the directory that contains your templates.

- Replace **<stack>** with the name of the stack for which the network VIPs are provisioned, for example, **overcloud**.

- Replace **<deployment_file>** with the name of the heat environment file to generate for inclusion in the deployment command, for example **/home/stack/templates/overcloud-vip-deployed.yaml**.

- Replace **<vip_definition_file>** with the name of your VIP definition file, for example, **vip_data.yaml**.

5. When the network VIP provisioning is complete, you can use the following commands to check the created VIPs:

```
(undercloud)$ openstack port list
(undercloud)$ openstack port show <port>
```

- Replace **<port>** with the name or UUID of the port you want to check.

Next steps

- Provisioning bare metal overcloud nodes

7.2. PROVISIONING BARE METAL OVERCLOUD NODES

To configure a Red Hat OpenStack Platform (RHOSP) environment, you must perform the following tasks:

1. Register the bare-metal nodes for your overcloud.

2. Provide director with an inventory of the hardware of the bare-metal nodes.

3. Configure the quantity, attributes, and network layout of the bare-metal nodes in a node definition file.

4. Assign each bare metal node a resource class that matches the node to its designated role.

You can also perform additional optional tasks, such as matching profiles to designate overcloud nodes.

7.2.1. Registering nodes for the overcloud

Director requires a node definition template that specifies the hardware and power management details of your nodes. You can create this template in JSON format, **nodes.json**, or YAML format, **nodes.yaml**.

Procedure

1. Create a template named **nodes.json** or **nodes.yaml** that lists your nodes. Use the following JSON and YAML template examples to understand how to structure your node definition template:

Example JSON template

```
{
  "nodes": [{
    "name": "node01",
    "ports": [{
      "address": "aa:aa:aa:aa:aa:aa",
      "physical_network": "ctlplane",
      "local_link_connection": {
        "switch_id": "52:54:00:00:00:00",
        "port_id": "p0"
      }
    }],
    "cpu": "4",
    "memory": "6144",
    "disk": "40",
    "arch": "x86_64",
    "pm_type": "ipmi",
    "pm_user": "admin",
    "pm_password": "p@55w0rd!",
    "pm_addr": "192.168.24.205"
  },
  {
    "name": "node02",
    "ports": [{
      "address": "bb:bb:bb:bb:bb:bb",
      "physical_network": "ctlplane",
      "local_link_connection": {
        "switch_id": "52:54:00:00:00:00",
        "port_id": "p0"
      }
    }],
    "cpu": "4",
    "memory": "6144",
    "disk": "40",
    "arch": "x86_64",
    "pm_type": "ipmi",
    "pm_user": "admin",
    "pm_password": "p@55w0rd!",
    "pm_addr": "192.168.24.206"
  }]
}
```

Example YAML template

```
nodes:
  - name: "node01"
    ports:
      - address: "aa:aa:aa:aa:aa:aa"
        physical_network: ctlplane
        local_link_connection:
          switch_id: 52:54:00:00:00:00
          port_id: p0
    cpu: 4
    memory: 6144
    disk: 40
    arch: "x86_64"
```

```
    pm_type: "ipmi"
    pm_user: "admin"
    pm_password: "p@55w0rd!"
    pm_addr: "192.168.24.205"
  - name: "node02"
    ports:
      - address: "bb:bb:bb:bb:bb:bb"
        physical_network: ctlplane
        local_link_connection:
          switch_id: 52:54:00:00:00:00
          port_id: p0
    cpu: 4
    memory: 6144
    disk: 40
    arch: "x86_64"
    pm_type: "ipmi"
    pm_user: "admin"
    pm_password: "p@55w0rd!"
    pm_addr: "192.168.24.206"
```

This template contains the following attributes:

name

The logical name for the node.

ports

The port to access the specific IPMI device. You can define the following optional port attributes:

- **address**: The MAC address for the network interface on the node. Use only the MAC address for the Provisioning NIC of each system.

- **physical_network**: The physical network that is connected to the Provisioning NIC.

- **local_link_connection**: If you use IPv6 provisioning and LLDP does not correctly populate the local link connection during introspection, you must include fake data with the **switch_id** and **port_id** fields in the **local_link_connection** parameter. For more information on how to include fake data, see Using director introspection to collect bare metal node hardware information.

cpu

(Optional) The number of CPUs on the node.

memory

(Optional) The amount of memory in MB.

disk

(Optional) The size of the hard disk in GB.

arch

(Optional) The system architecture.

pm_type

The power management driver that you want to use. This example uses the IPMI driver (**ipmi**).

> **NOTE**
>
> IPMI is the preferred supported power management driver. For more information about supported power management types and their options, see Power management drivers. If these power management drivers do not work as expected, use IPMI for your power management.

pm_user; pm_password

The IPMI username and password.

pm_addr

The IP address of the IPMI device.

2. Verify the template formatting and syntax:

```
$ source ~/stackrc
(undercloud)$ openstack overcloud node import --validate-only ~/nodes.json
```

3. Save the template file to the home directory of the **stack** user (**/home/stack/nodes.json**).

4. Import the template to director to register each node from the template into director:

```
(undercloud)$ openstack overcloud node import ~/nodes.json
```

5. Wait for the node registration and configuration to complete. When complete, confirm that director has successfully registered the nodes:

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

7.2.2. Creating an inventory of the bare-metal node hardware

Director needs the hardware inventory of the nodes in your Red Hat OpenStack Platform (RHOSP) deployment for profile tagging, benchmarking, and manual root disk assignment.

You can provide the hardware inventory to director by using one of the following methods:

- Automatic: You can use director's introspection process, which collects the hardware information from each node. This process boots an introspection agent on each node. The introspection agent collects hardware data from the node and sends the data back to director. Director stores the hardware data in the OpenStack internal database.

- Manual: You can manually configure a basic hardware inventory for each bare metal machine. This inventory is stored in the Bare Metal Provisioning service (ironic) and is used to manage and deploy the bare-metal machines.

The director automatic introspection process provides the following advantages over the manual method for setting the Bare Metal Provisioning service ports:

- Introspection records all of the connected ports in the hardware information, including the port to use for PXE boot if it is not already configured in **nodes.yaml**.

- Introspection sets the **local_link_connection** attribute for each port if the attribute is discoverable using LLDP. When you use the manual method, you must configure **local_link_connection** for each port when you register the nodes.

- Introspection sets the **physical_network** attribute for the Bare Metal Provisioning service ports when deploying a spine-and-leaf or DCN architecture.

7.2.2.1. Using director introspection to collect bare metal node hardware information

After you register a physical machine as a bare metal node, you can automatically add its hardware details and create ports for each of its Ethernet MAC addresses by using director introspection.

TIP

As an alternative to automatic introspection, you can manually provide director with the hardware information for your bare metal nodes. For more information, see Manually configuring bare metal node hardware information.

Prerequisites

- You have registered the bare-metal nodes for your overcloud.

Procedure

1. Log in to the undercloud host as the **stack** user.

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

   ```
   $ source ~/stackrc
   ```

3. Run the pre-introspection validation group to check the introspection requirements:

   ```
   (undercloud)$ validation run --group pre-introspection \
   --inventory <inventory_file>
   ```

   - Replace **<inventory_file>** with the name and location of the Ansible inventory file, for example, **~/tripleo-deploy/undercloud/tripleo-ansible-inventory.yaml**.

     NOTE

     When you run a validation, the **Reasons** column in the output is limited to 79 characters. To view the validation result in full, view the validation log files.

4. Review the results of the validation report.

5. Optional: Review detailed output from a specific validation:

   ```
   (undercloud)$ validation history get --full <UUID>
   ```

   - Replace **<UUID>** with the UUID of the specific validation from the report that you want to review.

     IMPORTANT

     A **FAILED** validation does not prevent you from deploying or running Red Hat OpenStack Platform. However, a **FAILED** validation can indicate a potential issue with a production environment.

6. Inspect the hardware attributes of each node. You can inspect the hardware attributes of all nodes, or specific nodes:

   - Inspect the hardware attributes of all nodes:

     ```
     (undercloud)$ openstack overcloud node introspect --all-manageable --provide
     ```

     - Use the **--all-manageable** option to introspect only the nodes that are in a managed state. In this example, all nodes are in a managed state.

     - Use the **--provide** option to reset all nodes to an **available** state after introspection.

   - Inspect the hardware attributes of specific nodes:

     ```
     (undercloud)$ openstack overcloud node introspect --provide <node1> [node2] [noden]
     ```

     - Use the **--provide** option to reset all the specified nodes to an **available** state after introspection.

     - Replace **<node1>**, **[node2]**, and all nodes up to **[noden]** with the UUID of each node that you want to introspect.

7. Monitor the introspection progress logs in a separate terminal window:

   ```
   (undercloud)$ sudo tail -f /var/log/containers/ironic-inspector/ironic-inspector.log
   ```

   > **IMPORTANT**
   >
   > Ensure that the introspection process runs to completion. Introspection usually takes 15 minutes for bare metal nodes. However, incorrectly sized introspection networks can cause it to take much longer, which can result in the introspection failing.

8. Optional: If you have configured your undercloud for bare metal provisioning over IPv6, then you need to also check that LLDP has set the **local_link_connection** for Bare Metal Provisioning service (ironic) ports:

   ```
   $ openstack baremetal port list --long -c UUID -c "Node UUID" -c "Local Link Connection"
   ```

   - If the Local Link Connection field is empty for the port on your bare metal node, you must populate the **local_link_connection** value manually with fake data. The following example sets the fake switch ID to **52:54:00:00:00:00**, and the fake port ID to **p0**:

     ```
     $ openstack baremetal port set <port_uuid> \
     --local-link-connection switch_id=52:54:00:00:00:00 \
     --local-link-connection port_id=p0
     ```

   - Verify that the Local Link Connection field contains the fake data:

     ```
     $ openstack baremetal port list --long -c UUID -c "Node UUID" -c "Local Link Connection"
     ```

After the introspection completes, all nodes change to an **available** state.

7.2.2.2. Manually configuring bare-metal node hardware information

After you register a physical machine as a bare metal node, you can manually add its hardware details and create bare-metal ports for each of its Ethernet MAC addresses. You must create at least one bare-metal port before deploying the overcloud.

TIP

As an alternative to manual introspection, you can use the automatic director introspection process to collect the hardware information for your bare metal nodes. For more information, see Using director introspection to collect bare metal node hardware information.

Prerequisites

- You have registered the bare-metal nodes for your overcloud.

- You have configured **local_link_connection** for each port on the registered nodes in **nodes.json**. For more information, see Registering nodes for the overcloud .

Procedure

1. Log in to the undercloud host as the **stack** user.

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

   ```
   $ source ~/stackrc
   ```

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

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

   - Replace **<node>** with the ID of the bare metal node.

   - Replace **<kernel_file>** with the path to the **.kernel** image, for example, **file:///var/lib/ironic/httpboot/agent.kernel**.

   - Replace **<initramfs_file>** with the path to the **.initramfs** image, for example, **file:///var/lib/ironic/httpboot/agent.ramdisk**.

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

   ```
   (undercloud)$ 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.

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

   ```
   (undercloud)$ 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.

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

   ```
   (undercloud)$ 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.

7. 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:

```
(undercloud)$ openstack baremetal port create --node <node_uuid> <mac_address>
```

- Replace **<node_uuid>** with the unique ID of the bare metal node.

- Replace **<mac_address>** with the MAC address of the NIC used to PXE boot.

8. Validate the configuration of the node:

```
(undercloud)$ openstack baremetal node validate <node>
---------------------------------------------------------------
| Interface  | Result | Reason                      |
---------------------------------------------------------------
| bios       | True   |                             |
| boot       | True   |                             |
| console    | True   |                             |
| deploy     | False  | Node 229f0c3d-354a-4dab-9a88-ebd318249ad6 |
|            |        | failed to validate deploy image info.     |
|            |        | Some parameters were missing. Missing are:|
|            |        | [instance_info.image_source]              |
| inspect    | True   |                             |
| management | True   |                             |
| network    | True   |                             |
| power      | True   |                             |
| raid       | True   |                             |
| rescue     | True   |                             |
| storage    | True   |                             |
---------------------------------------------------------------
```

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

- **False**: The interface has failed validation. If the reason provided includes missing the **instance_info.image_source** parameter, this might be because it is populated during provisioning, therefore it has 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.

7.2.3. Provisioning bare metal nodes for the overcloud

To provision your bare metal nodes, you define the quantity and attributes of the bare metal nodes that you want to deploy in a node definition file in YAML format, and assign overcloud roles to these nodes. You also define the network layout of the nodes.

The provisioning process creates a heat environment file from your node definition file. This heat environment file contains the node specifications you configured in your node definition file, including node count, predictive node placement, custom images, and custom NICs. When you deploy your overcloud, include this file in the deployment command. The provisioning process also provisions the port resources for all networks defined for each node or role in the node definition file.

Prerequisites

- The undercloud is installed. For more information, see Installing director .

- The bare metal nodes are registered, introspected, and available for provisioning and deployment. For more information, see Registering nodes for the overcloud and Creating an inventory of the bare metal node hardware.

Procedure

1. Source the **stackrc** undercloud credential file:

   ```
   $ source ~/stackrc
   ```

2. Create the **overcloud-baremetal-deploy.yaml** node definition file and define the node count for each role that you want to provision. For example, to provision three Controller nodes and three Compute nodes, add the following configuration to your **overcloud-baremetal-deploy.yaml** file:

   ```
   - name: Controller
     count: 3
   - name: Compute
     count: 3
   ```

3. Optional: Configure predictive node placements. For example, use the following configuration to provision three Controller nodes on nodes **node00**, **node01**, and **node02**, and three Compute nodes on **node04**, **node05**, and **node06**:

   ```
   - name: Controller
     count: 3
     instances:
     - hostname: overcloud-controller-0
       name: node00
     - hostname: overcloud-controller-1
       name: node01
     - hostname: overcloud-controller-2
       name: node02
   - name: Compute
     count: 3
     instances:
     - hostname: overcloud-novacompute-0
       name: node04
     - hostname: overcloud-novacompute-1
       name: node05
     - hostname: overcloud-novacompute-2
       name: node06
   ```

4. Optional: By default, the provisioning process uses the **overcloud-hardened-uefi-full.qcow2** image. You can change the image used on specific nodes, or the image used for all nodes for a role, by specifying the local or remote URL for the image. The following examples change the image to a local QCOW2 image:

   Specific nodes

   ```
   - name: Controller
     count: 3
   ```

```
instances:
- hostname: overcloud-controller-0
  name: node00
  image:
    href: file:///var/lib/ironic/images/overcloud-custom.qcow2
- hostname: overcloud-controller-1
  name: node01
  image:
    href: file:///var/lib/ironic/images/overcloud-full-custom.qcow2
- hostname: overcloud-controller-2
  name: node02
  image:
    href: file:///var/lib/ironic/images/overcloud-full-custom.qcow2
```

All nodes for a role

```
- name: Controller
  count: 3
  defaults:
    image:
      href: file:///var/lib/ironic/images/overcloud-custom.qcow2
  instances:
  - hostname: overcloud-controller-0
    name: node00
  - hostname: overcloud-controller-1
    name: node01
  - hostname: overcloud-controller-2
    name: node02
```

5. Define the network layout for all nodes for a role, or the network layout for specific nodes:

Specific nodes

The following example provisions the networks for a specific Controller node, and allocates a predictable IP to the node for the Internal API network:

```
- name: Controller
  count: 3
  defaults:
    network_config:
      template: /home/stack/templates/nic-config/myController.j2
      default_route_network:
      - external
    instances:
      - hostname: overcloud-controller-0
        name: node00
        networks:
          - network: ctlplane
            vif: true
          - network: external
            subnet: external_subnet
          - network: internal_api
            subnet: internal_api_subnet01
            fixed_ip: 172.21.11.100
          - network: storage
```

```
        subnet: storage_subnet01
      - network: storage_mgmt
        subnet: storage_mgmt_subnet01
      - network: tenant
        subnet: tenant_subnet01
```

All nodes for a role

The following example provisions the networks for the Controller and Compute roles:

```
- name: Controller
  count: 3
  defaults:
    networks:
    - network: ctlplane
      vif: true
    - network: external
      subnet: external_subnet
    - network: internal_api
      subnet: internal_api_subnet01
    - network: storage
      subnet: storage_subnet01
    - network: storage_mgmt
      subnet: storage_mgmt_subnet01
    - network: tenant
      subnet: tenant_subnet01
    network_config:
      template: /home/stack/templates/nic-config/myController.j2  ❶
      default_route_network:
      - external
- name: Compute
  count: 3
  defaults:
    networks:
    - network: ctlplane
      vif: true
    - network: internal_api
      subnet: internal_api_subnet02
    - network: tenant
      subnet: tenant_subnet02
    - network: storage
      subnet: storage_subnet02
    network_config:
      template: /home/stack/templates/nic-config/myCompute.j2
```

❶ You can use the example NIC templates located in **/usr/share/ansible/roles/tripleo_network_config/templates** to create your own NIC templates in your local environment file directory.

6. Optional: Configure the disk partition size allocations if the default disk partition sizes do not meet your requirements. For example, the default partition size for the **/var/log** partition is 10 GB. Consider your log storage and retention requirements to determine if 10 GB meets your requirements. If you need to increase the allocated disk size for your log storage, add the following configuration to your node definition file to override the defaults:

```
ansible_playbooks:
 - playbook: /usr/share/ansible/tripleo-playbooks/cli-overcloud-node-growvols.yaml
   extra_vars:
    role_growvols_args:
     default:
       /=8GB
       /tmp=1GB
       /var/log=<log_size>GB
       /var/log/audit=2GB
       /home=1GB
       /var=100%
```

- Replace **<log_size>** with the size of the disk to allocate to log files.

7. If you use the Object Storage service (swift) and the whole disk overcloud image, **overcloud-hardened-uefi-full**, you need to configure the size of the /**srv** partition based on the size of your disk and your storage requirements for /**var** and /**srv**. For more information, see Configuring whole disk partitions for the Object Storage service.

8. Optional: Designate the overcloud nodes for specific roles by using custom resource classes or the profile capability. For more information, see Designating overcloud nodes for roles by matching resource classes and Designating overcloud nodes for roles by matching profiles .

9. Define any other attributes that you want to assign to your nodes. For more information about the properties you can use to configure node attributes in your node definition file, see Bare metal node provisioning attributes. For an example node definition file, see Example node definition file.

10. Provision the overcloud nodes:

```
(undercloud)$ openstack overcloud node provision \
 [--templates <templates_directory> \]
 --stack <stack> \
 --network-config \
 --output <deployment_file> \
 /home/stack/templates/<node_definition_file>
```

- Optional: Include the **--templates** option to use your own templates instead of the default templates located in /**usr**/**share**/**openstack-tripleo-heat-templates**. Replace **<templates_directory>** with the path to the directory that contains your templates.

- Replace **<stack>** with the name of the stack for which the bare-metal nodes are provisioned. If not specified, the default is **overcloud**.

- Include the **--network-config** optional argument to provide the network definitions to the **cli-overcloud-node-network-config.yaml** Ansible playbook. The **cli-overcloud-node-network-config.yaml** playbook uses the **os-net-config** tool to apply the network configuration on the deployed nodes. If you do not use **--network-config** to provide the network definitions, then you must configure the **{{role.name}}NetworkConfigTemplate** parameters in your **network-environment.yaml** file, otherwise the default network definitions are used.

- Replace **<deployment_file>** with the name of the heat environment file to generate for inclusion in the deployment command, for example /**home**/**stack**/**templates**/**overcloud-baremetal-deployed.yaml**.

- Replace **<node_definition_file>** with the name of your node definition file, for example, **overcloud-baremetal-deploy.yaml**.

11. Monitor the provisioning progress in a separate terminal:

    (undercloud)$ watch openstack baremetal node list

    - When provisioning is successful, the node state changes from **available** to **active**.

    - If the node provisioning fails because of a node hardware or network configuration failure, then you can remove the failed node before running the provisioning step again. For more information, see Removing failed bare-metal nodes from the node definition file .

12. Use the **metalsmith** tool to obtain a unified view of your nodes, including allocations and ports:

    (undercloud)$ metalsmith list

13. Verify the association of nodes to hostnames:

    (undercloud)$ openstack baremetal allocation list

Next steps

- Configuring and deploying the overcloud

7.2.4. Bare-metal node provisioning attributes

Use the following tables to understand the available properties for configuring node attributes, and the values that are available for you to use when you provision bare-metal nodes with the **openstack baremetal node provision** command.

- Role properties: Use the role properties to define each role.

- Default and instance properties for each role: Use the default or instance properties to specify the selection criteria for allocating nodes from the pool of available nodes, and to set attributes and network configuration properties on the bare-metal nodes.

For information on creating baremetal definition files, see Provisioning bare metal nodes for the overcloud.

Table 7.1. Role properties

Property	Value
name	(Mandatory) Role name.
count	The number of nodes that you want to provision for this role. The default value is **1**.
defaults	A dictionary of default values for **instances** entry properties. An **instances** entry property overrides any defaults that you specify in the **defaults** parameter.

Property	Value
instances	A dictionary of values that you can use to specify attributes for specific nodes. For more information about supported properties in the **instances** parameter, see **defaults** and **instances** properties. The number of nodes defined must not be greater than the value of the **count** parameter.
hostname_format	Overrides the default hostname format for this role. The default generated hostname is derived from the overcloud stack name, the role, and an incrementing index, all in lower case. For example, the default format for the Controller role is **%stackname%-controller-%index%**. Only the Compute role does not follow the role name rule. The Compute default format is **%stackname%-novacompute-%index%**.
ansible_playbooks	A dictionary of values for Ansible playbooks and Ansible variables. The playbooks are run against the role instances after node provisioning, prior to the node network configuration. For more information about specifying Ansible playbooks, see **ansible_playbooks** properties.

Table 7.2. **defaults** and **instances** properties

Property	Value
hostname	(**instances** only) Specifies the hostname of the node that the **instance** properties apply to. The hostname is derived from the **hostname_format** property. You can use custom hostnames.
name	(**instances** only) The name of the node that you want to provision.
image	Details of the image that you want to provision onto the node. For information about supported **image** properties, see **image** properties.
capabilities	Selection criteria to match the node capabilities.
config_drive	Add data and first-boot commands to the config-drive passed to the node. For more information, see **config_drive** properties. NOTE Only use **config_drive** for configuration that must be performed on first boot. For all other custom configurations, create an Ansible playbook and use the **ansible_playbooks** property to execute the playbook against the role instances after node provisioning.
managed	Set to **true** (default) to provision the instance with metalsmith. Set to **false** to handle the instance as pre-provisioned.

Property	Value
networks	List of dictionaries that represent instance networks. For more information about configuring network attributes, see **network** properties.
network_config	Link to the network configuration file for the role or instance. For more information about configuring the link to the network configuration file, see **network_config** properties.
profile	Selection criteria to for profile matching. For more information, see Designating overcloud nodes for roles by matching profiles
provisioned	Set to **true** (default) to provision the node. Set to **false** to unprovision a node. For more information, see Scaling down bare-metal nodes.
resource_class	Selection criteria to match the resource class of the node. The default value is **baremetal**. For more information, see Designating overcloud nodes for roles by matching resource classes.
root_size_gb	Size of the root partition in GiB. The default value is **49**.
swap_size_mb	Size of the swap partition in MiB.
traits	A list of traits as selection criteria to match the node traits.

Table 7.3. **image** properties

Property	Value
href	Specifies the URL of the root partition or whole disk image that you want to provision onto the node. Supported URL schemes: **file://**, **http://**, and **https://**. NOTE If you use the **file://** URL scheme to specify a local URL for the image then the image path must point to the **/var/lib/ironic/images/** directory, because **/var/lib/ironic/images** is bind-mounted from the undercloud into the **ironic-conductor** container explicitly for serving images.
checksum	Specifies the MD5 checksum of the root partition or whole disk image. Required when the **href** is a URL.
kernel	Specifies the image reference or URL of the kernel image. Use this property only for partition images.

Property	Value
ramdisk	Specifies the image reference or URL of the ramdisk image. Use this property only for partition images.

Table 7.4. **network** properties

Property	Value
fixed_ip	The specific IP address that you want to use for this network.
network	The network where you want to create the network port.
subnet	The subnet where you want to create the network port.
port	Existing port to use instead of creating a new port.
vif	Set to **true** on the provisioning network (**ctlplane**) to attach the network as a virtual interface (VIF). Set to **false** to create the Networking service API resource without a VIF attachment.

Table 7.5. **network_config** properties

Property	Value
template	Specifies the Ansible J2 NIC configuration template to use when applying node network configuration. For information on configuring the NIC template, see Configuring overcloud networking.
physical_bridge_name	The name of the OVS bridge to create for accessing external networks. The default bridge name is **br-ex**.
public_interface_name	Specifies the name of the interface to add to the public bridge. The default interface is **nic1**.
network_config_update	Set to **true** to apply network configuration changes on update. Disabled by default.
net_config_data_lookup	Specifies the NIC mapping configuration, **os-net-config**, for each node or node group.
default_route_network	The network to use for the default route. The default route network is **ctlplane**
networks_skip_config	List of networks to skip when configuring the node networking.

Property	Value
dns_search_domains	A list of DNS search domains to be added to **resolv.conf**, in order of priority.
bond_interface_ovs_opti ons	The OVS options or bonding options to use for the bond interface, for example, **lacp=active** and **bond_mode=balance-slb** for OVS bonds, and **mode=4** for Linux bonds.
num_dpdk_interface_rx_ queues	Specifies the number of required RX queues for DPDK bonds or DPDK ports.

Table 7.6. **config_drive** properties

Property	Value

cloud_config	Dictionary of **cloud-init** cloud configuration data for tasks to run on node boot. For example, to write a custom name server to the **resolve.conf** file on first boot, add the following **cloud_config** to your **config_drive** property:
	
	<pre>config_drive:
	  cloud_config:
	    manage_resolv_conf: true
	    resolv_conf:
	      nameservers:
	        - 8.8.8.8
	        - 8.8.4.4
	      searchdomains:
	        - abc.example.com
	        - xyz.example.com
	      domain: example.com
	      sortlist:
	        - 10.0.0.1/255
	        - 10.0.0.2
	      options:
	        rotate: true
	        timeout: 1</pre>

meta_data	Extra metadata to include with the config-drive **cloud-init** metadata. The metadata is added to the generated metadata set on the role name: **public_keys**, **uuid**, **name**, **hostname**, and **instance-type**. **Cloud-init** makes this metadata available as instance data.

Table 7.7. **ansible_playbooks** properties

Property	Value
playbook	The path to the Ansible playbook, relative to the roles definition YAML file.

Property	Value

extra_vars	Extra Ansible variables to set when running the playbook. Use the following syntax to specify extra variables:
	
	```\nansible_playbooks:\n - playbook: a_playbook.yaml\n extra_vars:\n param1: value1\n param2: value2\n```
	
	For example, to grow the LVM volumes of any node deployed with the whole disk overcloud image **overcloud-hardened-uefi-full.qcow2**, add the following extra variable to your playbook property:
	
	```\nansible_playbooks:\n - playbook: /usr/share/ansible/tripleo-playbooks/cli-overcloud-node-growvols.yaml\n extra_vars:\n role_growvols_args:\n default:\n /=8GB\n /tmp=1GB\n /var/log=10GB\n /var/log/audit=2GB\n /home=1GB\n /var=100%\n Controller:\n /=8GB\n /tmp=1GB\n /var/log=10GB\n /var/log/audit=2GB\n /home=1GB\n /srv=50GB\n /var=100%\n```

7.2.5. Removing failed bare-metal nodes from the node definition file

If the node provisioning fails because of a node hardware or network configuration failure, then you can remove the failed node before running the provisioning step again. To remove a bare-metal node that has failed during provisioning, tag the node that you want to remove from the stack in the node definition file, and unprovision the node before provisioning the working bare-metal nodes.

Prerequisites

- The undercloud is installed. For more information, see Installing director.

- The bare-metal node provisioning failed because of a node hardware failure.

Procedure

1. Source the **stackrc** undercloud credential file:

```
$ source ~/stackrc
```

2. Open your **overcloud-baremetal-deploy.yaml** node definition file.

3. Decrement the **count** parameter for the role that the node is allocated to. For example, the following configuration updates the count parameter for the **ObjectStorage** role to reflect that the number of nodes dedicated to **ObjectStorage** is reduced to 3:

```
- name: ObjectStorage
  count: 3
```

4. Define the **hostname** and **name** of the node that you want to remove from the stack, if it is not already defined in the **instances** attribute for the role.

5. Add the attribute **provisioned: false** to the node that you want to remove. For example, to remove the node **overcloud-objectstorage-1** from the stack, include the following snippet in your **overcloud-baremetal-deploy.yaml** file:

```
- name: ObjectStorage
  count: 3
  instances:
  - hostname: overcloud-objectstorage-0
    name: node00
  - hostname: overcloud-objectstorage-1
    name: node01
    # Removed from cluster due to disk failure
    provisioned: false
  - hostname: overcloud-objectstorage-2
    name: node02
  - hostname: overcloud-objectstorage-3
    name: node03
```

6. Unprovision the bare-metal nodes:

```
(undercloud)$ openstack overcloud node unprovision \
 --stack <stack> \
 --network-ports \
 /home/stack/templates/overcloud-baremetal-deploy.yaml
```

- Replace **<stack>** with the name of the stack for which the bare-metal nodes are provisioned. If not specified, the default is **overcloud**.

7. Provision the overcloud nodes to generate an updated heat environment file for inclusion in the deployment command:

```
(undercloud)$ openstack overcloud node provision \
--stack <stack> \
--output <deployment_file> \
/home/stack/templates/overcloud-baremetal-deploy.yaml
```

- Replace **<deployment_file>** with the name of the heat environment file to generate for inclusion in the deployment command, for example **/home/stack/templates/overcloud-baremetal-deployed.yaml**.

7.2.6. Designating overcloud nodes for roles by matching resource classes

You can designate overcloud nodes for specific roles by using custom resource classes. Resource classes match your nodes to deployment roles. By default all nodes are assigned the resource class of **baremetal**.

> **NOTE**
>
> To change the resource class assigned to a node after the node is provisioned you must use the scale down procedure to unprovision the node, then use the scale up procedure to reprovision the node with the new resource class assignment. For more information, see Scaling overcloud nodes .

Prerequisites

- You are performing the initial provisioning of your bare metal nodes for the overcloud.

Procedure

1. Assign each bare metal node that you want to designate for a role with a custom resource class:

   ```
   (undercloud)$ openstack baremetal node set \
   --resource-class <resource_class> <node>
   ```

   - Replace **<resource_class>** with a name for the resource class, for example, **baremetal.CPU-PINNING**.

   - Replace **<node>** with the ID of the bare metal node.

2. Add the role to your **overcloud-baremetal-deploy.yaml** file, if not already defined.

3. Specify the resource class that you want to assign to the nodes for the role:

   ```
   - name: <role>
     count: 1
     defaults:
       resource_class: <resource_class>
   ```

   - Replace **<role>** with the name of the role.

   - Replace **<resource_class>** with the name you specified for the resource class in step 1.

4. Return to Provisioning bare metal nodes for the overcloud to complete the provisioning process.

7.2.7. Designating overcloud nodes for roles by matching profiles

You can designate overcloud nodes for specific roles by using the profile capability. Profiles match node capabilities to deployment roles.

TIP

You can also perform automatic profile assignment by using introspection rules. For more information, see Configuring automatic profile tagging .

> **NOTE**
>
> To change the profile assigned to a node after the node is provisioned you must use the scale down procedure to unprovision the node, then use the scale up procedure to reprovision the node with the new profile assignment. For more information, see Scaling overcloud nodes.

Prerequisites

- You are performing the initial provisioning of your bare metal nodes for the overcloud.

Procedure

1. Existing node capabilities are overwritten each time you add a new node capability. Therefore, you must retrieve the existing capabilities of each registered node in order to set them again:

   ```
   $ openstack baremetal node show <node> \
    -f json -c properties | jq -r .properties.capabilities
   ```

2. Assign the profile capability to each bare metal node that you want to match to a role profile, by adding **profile:<profile>** to the existing capabilities of the node:

   ```
   (undercloud)$ openstack baremetal node set  <node> \
    --property capabilities="profile:<profile>,<capability_1>,...,<capability_n>"
   ```

   - Replace **<node>** with the name or ID of the bare metal node.

   - Replace **<profile>** with the name of the profile that matches the role profile.

   - Replace **<capability_1>**, and all capabilities up to **<capability_n>**, with each capability that you retrieved in step 1.

3. Add the role to your **overcloud-baremetal-deploy.yaml** file, if not already defined.

4. Define the profile that you want to assign to the nodes for the role:

   ```
   - name: <role>
     count: 1
     defaults:
       profile: <profile>
   ```

   - Replace **<role>** with the name of the role.

   - Replace **<profile>** with the name of the profile that matches the node capability.

5. Return to Provisioning bare metal nodes for the overcloud to complete the provisioning process.

7.2.8. Configuring whole disk partitions for the Object Storage service

The whole disk image, **overcloud-hardened-uefi-full**, is partitioned into separate volumes. By default, the **/var** partition of nodes deployed with the whole disk overcloud image is automatically increased until the disk is fully allocated. If you use the Object Storage service (swift), configure the size of the **/srv** partition based on the size of your disk and your storage requirements for **/var** and **/srv**.

Prerequisites

- You are performing the initial provisioning of your bare metal nodes for the overcloud.

Procedure

1. Configure the **/srv** and **/var** partitions by using **role_growvols_args** as an extra Ansible variable in the Ansible playbooks definition in your **overcloud-baremetal-deploy.yaml** node definition file. Set either **/srv** or **/var** to an absolute size in GB, and set the other to 100% to consume the remaining disk space.

    - The following example configuration sets **/srv** to an absolute size for the Object Storage service deployed on the Controller node, and **/var** to 100% to consume the remaining disk space:

        ```
        ansible_playbooks:
          - playbook: /usr/share/ansible/tripleo-playbooks/cli-overcloud-node-growvols.yaml
            extra_vars:
              role_growvols_args:
                default:
                  /=8GB
                  /tmp=1GB
                  /var/log=10GB
                  /var/log/audit=2GB
                  /home=1GB
                  /var=100%
                Controller:
                  /=8GB
                  /tmp=1GB
                  /var/log=10GB
                  /var/log/audit=2GB
                  /home=1GB
                  /srv=50GB
                  /var=100%
        ```

    - The following example configuration sets **/var** to an absolute size, and **/srv** to 100% to consume the remaining disk space of the Object Storage node for the Object Storage service:

        ```
        ansible_playbooks:
          - playbook: /usr/share/ansible/tripleo-playbooks/cli-overcloud-node-growvols.yaml
            extra_vars:
              role_growvols_args:
                default:
                  /=8GB
                  /tmp=1GB
                  /var/log=10GB
                  /var/log/audit=2GB
                  /home=1GB
                  /var=100%
                ObjectStorage:
                  /=8GB
                  /tmp=1GB
                  /var/log=10GB
                  /var/log/audit=2GB
        ```

```
        /home=1GB
        /var=10GB
        /srv=100%
```

2. Return to Provisioning bare metal nodes for the overcloud to complete the provisioning process.

7.2.9. Example node definition file

The following example node definition file defines predictive node placements for three Controller nodes and three Compute nodes, and the default networks they use. The example also illustrates how to define custom roles that have nodes designated based on matching a resource class or a node capability profile.

```
- name: Controller
  count: 3
  defaults:
    image:
      href: file:///var/lib/ironic/images/overcloud-custom.qcow2
    networks:
    - network: ctlplane
      vif: true
    - network: external
      subnet: external_subnet
    - network: internal_api
      subnet: internal_api_subnet01
    - network: storage
      subnet: storage_subnet01
    - network: storagemgmt
      subnet: storage_mgmt_subnet01
    - network: tenant
      subnet: tenant_subnet01
    network_config:
        template: /home/stack/templates/nic-config/myController.j2
        default_route_network:
        - external
    profile: nodeCapability
  instances:
  - hostname: overcloud-controller-0
    name: node00
  - hostname: overcloud-controller-1
    name: node01
  - hostname: overcloud-controller-2
    name: node02
- name: Compute
  count: 3
  defaults:
    networks:
    - network: ctlplane
      vif: true
    - network: internal_api
      subnet: internal_api_subnet02
    - network: tenant
      subnet: tenant_subnet02
    - network: storage
```

```
      subnet: storage_subnet02
    network_config:
      template: /home/stack/templates/nic-config/myCompute.j2
    resource_class: baremetal.COMPUTE
  instances:
  - hostname: overcloud-novacompute-0
    name: node04
  - hostname: overcloud-novacompute-1
    name: node05
  - hostname: overcloud-novacompute-2
    name: node06
```

7.2.10. Enabling 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.

To boot a node with the **redfish** hardware type over virtual media, set the boot interface to **redfish-virtual-media** and 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 network for cleaning and provisioning.

Procedure

1. Log in to the undercloud host as the **stack** user.

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

```
$ source ~/stackrc
```

3. Set the Bare Metal Provisioning service boot interface to **redfish-virtual-media**:

```
(undercloud)$ openstack baremetal node set --boot-interface redfish-virtual-media <node>
```

- Replace **<node>** with the name of the node.

4. Define the ESP image:

```
(undercloud)$ openstack baremetal node set --driver-info bootloader=<esp> <node>
```

- Replace **<esp>** with the Image service (glance) image UUID or the URL for the ESP image.

- Replace **<node>** with the name of the node.

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

```
(undercloud)$ openstack baremetal port create --pxe-enabled True \
--node <node_uuid> <mac_address>
```

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

- Replace **<mac_address>** with the MAC address of the NIC on the bare-metal node.

7.2.11. Defining the root disk for multi-disk Ceph clusters

Ceph Storage nodes typically use multiple disks. Director must identify the root disk in multiple disk configurations. The overcloud image is written to the root disk during the provisioning process.

Hardware properties are used to identify the root disk. For more information about properties you can use to identify the root disk, see Properties that identify the root disk .

Procedure

1. Verify the disk information from the hardware introspection of each node:

```
(undercloud)$ openstack baremetal introspection data save <node_uuid> --file
<output_file_name>
```

- Replace **<node_uuid>** with the UUID of the node.

- Replace **<output_file_name>** with the name of the file that contains the output of the node introspection.
  For example, the data for one node might show three disks:

```
[
  {
    "size": 299439751168,
    "rotational": true,
    "vendor": "DELL",
    "name": "/dev/sda",
    "wwn_vendor_extension": "0x1ea4dcc412a9632b",
```

```
      "wwn_with_extension": "0x61866da04f3807001ea4dcc412a9632b",
      "model": "PERC H330 Mini",
      "wwn": "0x61866da04f380700",
      "serial": "61866da04f3807001ea4dcc412a9632b"
    }
    {
      "size": 299439751168,
      "rotational": true,
      "vendor": "DELL",
      "name": "/dev/sdb",
      "wwn_vendor_extension": "0x1ea4e13c12e36ad6",
      "wwn_with_extension": "0x61866da04f380d001ea4e13c12e36ad6",
      "model": "PERC H330 Mini",
      "wwn": "0x61866da04f380d00",
      "serial": "61866da04f380d001ea4e13c12e36ad6"
    }
    {
      "size": 299439751168,
      "rotational": true,
      "vendor": "DELL",
      "name": "/dev/sdc",
      "wwn_vendor_extension": "0x1ea4e31e121cfb45",
      "wwn_with_extension": "0x61866da04f37fc001ea4e31e121cfb45",
      "model": "PERC H330 Mini",
      "wwn": "0x61866da04f37fc00",
      "serial": "61866da04f37fc001ea4e31e121cfb45"
    }
  ]
```

2. Set the root disk for the node by using a unique hardware property:
   **(undercloud)$ openstack baremetal node set --property root_device='{<property_value>}'
   <node-uuid>**

   - Replace **<property_value>** with the unique hardware property value from the introspection data to use to set the root disk.

   - Replace **<node_uuid>** with the UUID of the node.

   > **NOTE**
   >
   > A unique hardware property is any property from the hardware introspection step that uniquely identifies the disk. For example, the following command uses the disk serial number to set the root disk:
   >
   > **(undercloud)$ openstack baremetal node set --property
   > root_device='{"serial": "61866da04f380d001ea4e13c12e36ad6"}'
   > 1a4e30da-b6dc-499d-ba87-0bd8a3819bc0**

3. Configure the BIOS of each node to first boot from the network and then the root disk.

Director identifies the specific disk to use as the root disk. When you run the **openstack overcloud node provision** command, director provisions and writes the overcloud image to the root disk.

7.2.12. Properties that identify the root disk

There are several properties that you can define to help director identify the root disk:

- **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.

> **IMPORTANT**
>
> Use the **name** property for devices with persistent names. Do not use the **name** property to set the root disk for devices that do not have persistent names because the value can change when the node boots.

7.2.13. Using the overcloud-minimal image to avoid using a Red Hat subscription entitlement

The default image for a Red Hat OpenStack Platform (RHOSP) deployment is **overcloud-hardened-uefi-full.qcow2**. The **overcloud-hardened-uefi-full.qcow2** image uses a valid Red Hat OpenStack Platform (RHOSP) subscription. You can use the **overcloud-minimal** image when you do not want to consume your subscription entitlements, to avoid reaching the limit of your paid Red Hat subscriptions. This is useful, for example, when you want to provision nodes with only Ceph daemons, or when you want to provision a bare operating system (OS) where you do not want to run any other OpenStack services. For information about how to obtain the **overcloud-minimal** image, see Obtaining images for overcloud nodes.

> **NOTE**
>
> The **overcloud-minimal** image supports only standard Linux bridges. The **overcloud-minimal** image does not support Open vSwitch (OVS) because OVS is an OpenStack service that requires a Red Hat OpenStack Platform subscription entitlement. OVS is not required to deploy Ceph Storage nodes. Use **linux_bond** instead of **ovs_bond** to define bonds. For more information about **linux_bond**, see Creating Linux bonds.

Procedure

1. Open your **overcloud-baremetal-deploy.yaml** file.

2. Add or update the **image** property for the nodes that you want to use the **overcloud-minimal** image. You can set the image to **overcloud-minimal** on specific nodes, or for all nodes for a role:

Specific nodes

```
- name: Ceph
  count: 3
  instances:
  - hostname: overcloud-ceph-0
    name: node00
    image:
      href: file:///var/lib/ironic/images/overcloud-minimal.qcow2
  - hostname: overcloud-ceph-1
    name: node01
    image:
      href: file:///var/lib/ironic/images/overcloud-full-custom.qcow2
  - hostname: overcloud-ceph-2
    name: node02
    image:
      href: file:///var/lib/ironic/images/overcloud-full-custom.qcow2
```

All nodes for a role

```
- name: Ceph
  count: 3
  defaults:
    image:
      href: file:///var/lib/ironic/images/overcloud-minimal.qcow2
  instances:
  - hostname: overcloud-ceph-0
    name: node00
  - hostname: overcloud-ceph-1
    name: node01
  - hostname: overcloud-ceph-2
    name: node02
```

3. In the **roles_data.yaml** role definition file, set the **rhsm_enforce** parameter to **False**.

```
rhsm_enforce: False
```

4. Run the provisioning command:

```
(undercloud)$ openstack overcloud node provision \
--stack stack \
--output /home/stack/templates/overcloud-baremetal-deployed.yaml \
/home/stack/templates/overcloud-baremetal-deploy.yaml
```

5. Pass the **overcloud-baremetal-deployed.yaml** environment file to the **openstack overcloud deploy** command.

7.3. CONFIGURING AND DEPLOYING THE OVERCLOUD

After you have provisioned the network resources and bare-metal nodes for your overcloud, you can configure your overcloud by using the unedited heat template files provided with your director installation, and any custom environment files you create. When you have completed the configuration of your overcloud, you can deploy the overcloud environment.

IMPORTANT

A basic overcloud uses local LVM storage for block storage, which is not a supported configuration. Red Hat recommends that you use an external storage solution, such as Red Hat Ceph Storage, for block storage.

7.3.1. Prerequisites

- You have provisioned the network resources and bare-metal nodes required for your overcloud.

7.3.2. Configuring your overcloud by using environment files

The undercloud includes a set of heat templates that form the plan for your overcloud creation. You can customize aspects of the overcloud with environment files, which are YAML-formatted files that override parameters and resources in the core heat template collection. You can include as many environment files as necessary. The environment file extension must be **.yaml** or **.template**.

Red Hat recommends that you organize your custom environment files in a separate directory, such as the **templates** directory.

You include environment files in your overcloud deployment by using the **-e** option. Any environment files that you add to the overcloud using the **-e** option become part of the stack definition of the overcloud. The order of the environment files is important because the parameters and resources that you define in subsequent environment files take precedence.

To modify the overcloud configuration after initial deployment, perform the following actions:

1. Modify parameters in the custom environment files and heat templates.

2. Run the **openstack overcloud deploy** command again with the same environment files.

Do not edit the overcloud configuration directly because director overrides any manual configuration when you update the overcloud stack.

NOTE

Open Virtual Networking (OVN) is the default networking mechanism driver in Red Hat OpenStack Platform 17.1. If you want to use OVN with distributed virtual routing (DVR), you must include the **environments/services/neutron-ovn-dvr-ha.yaml** file in the **openstack overcloud deploy** command. If you want to use OVN without DVR, you must include the **environments/services/neutron-ovn-ha.yaml** file in the **openstack overcloud deploy** command.

7.3.3. Creating an environment file for undercloud CA trust

If your undercloud uses TLS and the Certificate Authority (CA) is not publicly trusted, you can use the CA for SSL endpoint encryption that the undercloud operates. To ensure that the undercloud endpoints are accessible to the rest of your deployment, configure your overcloud nodes to trust the undercloud CA.

NOTE

For this approach to work, your overcloud nodes must have a network route to the public endpoint on the undercloud. It is likely that you must apply this configuration for deployments that rely on spine-leaf networking.

There are two types of custom certificates you can use in the undercloud:

- **User-provided certificates** - This definition applies when you have provided your own certificate. This can be from your own CA, or it can be self-signed. This is passed using the **undercloud_service_certificate** option. In this case, you must either trust the self-signed certificate, or the CA (depending on your deployment).

- **Auto-generated certificates** - This definition applies when you use **certmonger** to generate the certificate using its own local CA. Enable auto-generated certificates with the **generate_service_certificate** option in the **undercloud.conf** file. In this case, director generates a CA certificate at **/etc/pki/ca-trust/source/anchors/cm-local-ca.pem** and the director configures the undercloud's HAProxy instance to use a server certificate. Add the CA certificate to the **inject-trust-anchor-hiera.yaml** file to present the certificate to OpenStack Platform.

This example uses a self-signed certificate located in **/home/stack/ca.crt.pem**. If you use auto-generated certificates, use **/etc/pki/ca-trust/source/anchors/cm-local-ca.pem** instead.

Procedure

1. Open the certificate file and copy only the certificate portion. Do not include the key:

   ```
   $ vi /home/stack/ca.crt.pem
   ```

   The certificate portion you need looks similar to this shortened example:

   ```
   -----BEGIN CERTIFICATE-----
   MIIDlTCCAn2gAwIBAgIJAOnPtx2hHEhrMA0GCSqGSIb3DQEBCwUAMGExCzAJBgNV
   BAYTAlVTMQswCQYDVQQIDAJOQzEQMA4GA1UEBwwHUmFsZWlnaDEQMA4GA1UECg
   wH
   UmVkIEhhdDELMAkGA1UECwwCUUUxFDASBgNVBAMMCzE5Mi4xNjguMC4yMB4XDTE3
   -----END CERTIFICATE-----
   ```

2. Create a new YAML file called **/home/stack/inject-trust-anchor-hiera.yaml** with the following contents, and include the certificate you copied from the PEM file:

   ```
   parameter_defaults:
     CAMap:
       undercloud-ca:
         content: |
           -----BEGIN CERTIFICATE-----
           MIIDlTCCAn2gAwIBAgIJAOnPtx2hHEhrMA0GCSqGSIb3DQEBCwUAMGExCzAJBgNV

   BAYTAlVTMQswCQYDVQQIDAJOQzEQMA4GA1UEBwwHUmFsZWlnaDEQMA4GA1UECg
   wH

   UmVkIEhhdDELMAkGA1UECwwCUUUxFDASBgNVBAMMCzE5Mi4xNjguMC4yMB4XDTE3
           -----END CERTIFICATE-----
   ```

NOTE

- The certificate string must follow the PEM format.

- The **CAMap** parameter might contain other certificates relevant to SSL/TLS configuration.

3. Add the **/home/stack/inject-trust-anchor-hiera.yaml** file to your deployment command. Director copies the CA certificate to each overcloud node during the overcloud deployment. As a result, each node trusts the encryption presented by the undercloud's SSL endpoints.

7.3.4. Disabling TSX on new deployments

From Red Hat Enterprise Linux 8.3 onwards, the kernel disables support for the Intel Transactional Synchronization Extensions (TSX) feature by default.

You must explicitly disable TSX for new overclouds unless you strictly require it for your workloads or third party vendors.

Set the **KernelArgs** heat parameter in an environment file.

```
parameter_defaults:
  ComputeParameters:
    KernelArgs: "tsx=off"
```

Include the environment file when you run your **openstack overcloud deploy** command.

Additional resources

- "Guidance on Intel TSX impact on OpenStack guests (applies for RHEL 8.3 and above)"

7.3.5. Validating your overcloud configuration

Before deploying your overcloud, validate your heat templates and environment files.

IMPORTANT

- As a result of a change to the API in 17.0, the following validations are currently unstable:

  - switch-vlans

  - network-environment

  - dhcp-provisioning

- A **FAILED** validation does not prevent you from deploying or running Red Hat OpenStack Platform. However, a **FAILED** validation can indicate a potential issue with a production environment.

Procedure

1. Log in to the undercloud host as the **stack** user.

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

```
$ source ~/stackrc
```

3. Update your overcloud stack with all the environment files your deployment requires:

```
$ openstack overcloud deploy --templates \
  -e environment-file1.yaml \
  -e environment-file2.yaml \
  ...
  --stack-only
```

4. **Optional**: Create a YAML file that contains any validations that you want to exlcude from the validation run:

```
<validation_name>:
  hosts: <targeted_hostname>
```

- Replace **<validation_name>** with the name of the validation that you want to exlude from the validation run.

- Replace **<targeted_hostname>** with the name of the host that contains the validation.

5. Validate your overcloud stack:

```
$ validation run \
  --group pre-deployment \
  --inventory <inventory_file>
  [--skiplist <validation_skips>]
```

- Replace **<inventory_file>** with the name and location of the Ansible inventory file, for example, ~/**tripleo-deploy/undercloud/tripleo-ansible-inventory.yaml**.

- Replace **<validation_skips>** with the name and location of a YAML file that contains a list of validations to exclude from the validation run.
  NOTE:

- The **validation run --group pre-deployment** command includes the **node-disks** validation. Due to a known issue, this validation currently fails. To avoid this issue, add the **--skiplist** argument with a **yaml** file containing the **node-disks** validation to the **validation run -- group pre-deployment** command.

- When you run a validation, the **Reasons** column in the output is limited to 79 characters. To view the validation result in full, view the validation log files.

6. Review the results of the validation report:

```
$ validation history get [--full] [--validation-log-dir <log_dir>] <uuid>
```

- Optional: Use the **--full** option to view detailed output from the validation run.

- Optional: Use the **--validation-log-dir** option to write the output from the validation run to the validation logs.

- Replace **<uuid>** with the UUID of the validation run.

7.3.6. Creating your overcloud

The final stage in creating your Red Hat OpenStack Platform (RHOSP) overcloud environment is to run the **openstack overcloud deploy** command to create the overcloud. For information about the options available to use with the **openstack overcloud deploy** command, see Deployment command options.

Procedure

1. Collate the environment files and configuration files that you need for your overcloud environment, both the unedited heat template files provided with your director installation, and the custom environment files you created. This should include the following files:

   - **overcloud-baremetal-deployed.yaml** node definition file.

   - **overcloud-networks-deployed.yaml** network definition file.

   - **overcloud-vip-deployed.yaml** network VIP definition file.

   - The location of the container images for containerized OpenStack services.

   - Any environment files for Red Hat CDN or Satellite registration.

   - Any other custom environment files.

2. Organize the environment files and configuration files by their order of precedence, listing unedited heat template files first, followed by your environment files that contain custom configuration, such as overrides to the default properties.

3. Construct your **openstack overcloud deploy** command, specifying the configuration files and templates in the required order, for example:

   ```
   (undercloud) $ openstack overcloud deploy --templates \
    [-n /home/stack/templates/network_data.yaml \ ]
     -e /home/stack/templates/overcloud-baremetal-deployed.yaml\
     -e /home/stack/templates/overcloud-networks-deployed.yaml\
     -e /home/stack/templates/overcloud-vip-deployed.yaml \
     -e /home/stack/containers-prepare-parameter.yaml \
     -e /home/stack/inject-trust-anchor-hiera.yaml \
    [-r /home/stack/templates/roles_data.yaml ]
   ```

   -n /home/stack/templates/network_data.yaml

   Specifies the custom network configuration. Required if you use network isolation or custom composable networks. For information on configuring overcloud networks, see Configuring overcloud networking.

   -e /home/stack/containers-prepare-parameter.yaml

   Adds the container image preparation environment file. You generated this file during the undercloud installation and can use the same file for your overcloud creation.

   -e /home/stack/inject-trust-anchor-hiera.yaml

   Adds an environment file to install a custom certificate in the undercloud.

   -r /home/stack/templates/roles_data.yaml

   The generated roles data, if you use custom roles or want to enable a multi-architecture cloud.

4. When the overcloud creation completes, director provides a recap of the Ansible plays that were executed to configure the overcloud:

```
PLAY RECAP *********************************************************
overcloud-compute-0     : ok=160  changed=67  unreachable=0   failed=0
overcloud-controller-0 : ok=210  changed=93  unreachable=0   failed=0
undercloud              : ok=10   changed=7   unreachable=0   failed=0

Tuesday 15 October 2018  18:30:57 +1000 (0:00:00.107) 1:06:37.514 ******
=============================================================================
```

5. When the overcloud creation completes, director provides details to access your overcloud:

```
Ansible passed.
Overcloud configuration completed.
Overcloud Endpoint: http://192.168.24.113:5000
Overcloud Horizon Dashboard URL: http://192.168.24.113:80/dashboard
Overcloud rc file: /home/stack/overcloudrc
Overcloud Deployed
```

TIP

You can keep your deployment command in a file that you add to every time you update your configuration with a new env file.

7.3.7. Deployment command options

The following table lists the additional parameters for the **openstack overcloud deploy** command.

IMPORTANT

Some options are available in this release as a *Technology Preview* and therefore are not fully supported by Red Hat. They should only be used for testing and should not be used in a production environment. For more information about Technology Preview features, see Scope of Coverage Details.

Table 7.8. Deployment command options

Parameter	Description
--templates [TEMPLATES]	The directory that contains the heat templates that you want to deploy. If blank, the deployment command uses the default template location at **/usr/share/openstack-tripleo-heat-templates/**
--stack STACK	The name of the stack that you want to create or update
-t [TIMEOUT], **--timeout [TIMEOUT]**	The deployment timeout duration in minutes
--libvirt-type [LIBVIRT_TYPE]	The virtualization type that you want to use for hypervisors

Parameter	Description
--ntp-server [NTP_SERVER]	The Network Time Protocol (NTP) server that you want to use to synchronize time. You can also specify multiple NTP servers in a comma-separated list, for example: **--ntp-server 0.centos.pool.org,1.centos.pool.org**. For a high availability cluster deployment, it is essential that your Controller nodes are consistently referring to the same time source. Note that a typical environment might already have a designated NTP time source with established practices.
--no-proxy [NO_PROXY]	Defines custom values for the environment variable **no_proxy**, which excludes certain host names from proxy communication.
--overcloud-ssh-user OVERCLOUD_SSH_USER	Defines the SSH user to access the overcloud nodes. Normally SSH access occurs through the **tripleo-admin** user.
--overcloud-ssh-key OVERCLOUD_SSH_KEY	Defines the key path for SSH access to overcloud nodes.
--overcloud-ssh-network OVERCLOUD_SSH_NETWORK	Defines the network name that you want to use for SSH access to overcloud nodes.
-e [EXTRA HEAT TEMPLATE], **--environment-file [ENVIRONMENT FILE]**	Extra environment files that you want to pass to the overcloud deployment. You can specify this option more than once. Note that the order of environment files that you pass to the **openstack overcloud deploy** command is important. For example, parameters from each sequential environment file override the same parameters from earlier environment files.
--environment-directory	A directory that contains environment files that you want to include in deployment. The deployment command processes these environment files in numerical order, then alphabetical order.
-r ROLES_FILE	Defines the roles file and overrides the default **roles_data.yaml** in the **--templates** directory. The file location can be an absolute path or the path relative to **--templates**.
-n NETWORKS_FILE	Defines the networks file and overrides the default network_data.yaml in the **--templates** directory. The file location can be an absolute path or the path relative to **--templates**.

Parameter	Description
-p PLAN_ENVIRONMENT_FILE	Defines the plan Environment file and overrides the default **plan-environment.yaml** in the **--templates** directory. The file location can be an absolute path or the path relative to **--templates**.
--no-cleanup	Use this option if you do not want to delete temporary files after deployment, and log their location.
--update-plan-only	Use this option if you want to update the plan without performing the actual deployment.
--validation-errors-nonfatal	The overcloud creation process performs a set of pre-deployment checks. This option exits if any non-fatal errors occur from the pre-deployment checks. It is advisable to use this option as any errors can cause your deployment to fail.
--validation-warnings-fatal	The overcloud creation process performs a set of pre-deployment checks. This option exits if any non-critical warnings occur from the pre-deployment checks. openstack-tripleo-validations
--dry-run	Use this option if you want to perform a validation check on the overcloud without creating the overcloud.
--run-validations	Use this option to run external validations from the **openstack-tripleo-validations** package.
--skip-postconfig	Use this option to skip the overcloud post-deployment configuration.
--force-postconfig	Use this option to force the overcloud post-deployment configuration.
--skip-deploy-identifier	Use this option if you do not want the deployment command to generate a unique identifier for the **DeployIdentifier** parameter. The software configuration deployment steps only trigger if there is an actual change to the configuration. Use this option with caution and only if you are confident that you do not need to run the software configuration, such as scaling out certain roles.
--answers-file ANSWERS_FILE	The path to a YAML file with arguments and parameters.

Parameter	Description
--disable-password-generation	Use this option if you want to disable password generation for the overcloud services.
--deployed-server	Use this option if you want to deploy pre-provisioned overcloud nodes. Used in conjunction with **--disable-validations**.
--no-config-download, --stack-only	Use this option if you want to disable the **config-download** workflow and create only the stack and associated OpenStack resources. This command applies no software configuration to the overcloud.
--config-download-only	Use this option if you want to disable the overcloud stack creation and only run the **config-download** workflow to apply the software configuration.
--output-dir OUTPUT_DIR	The directory that you want to use for saved **config-download** output. The directory must be writeable by the mistral user. When not specified, director uses the default, which is **/var/lib/mistral/overcloud**.
--override-ansible-cfg OVERRIDE_ANSIBLE_CFG	The path to an Ansible configuration file. The configuration in the file overrides any configuration that **config-download** generates by default.
--config-download-timeout CONFIG_DOWNLOAD_TIMEOUT	The timeout duration in minutes that you want to use for **config-download** steps. If unset, director sets the default to the amount of time remaining from the **--timeout** parameter after the stack deployment operation.
--limit NODE1,NODE2	(Technology Preview) Use this option with a comma-separated list of nodes to limit the config-download playbook execution to a specific node or set of nodes. For example, the **--limit** option can be useful for scale-up operations, when you want to run config-download only on new nodes. This argument might cause live migration of instances between hosts to fail, see Running config-download with the ansible-playbook-command.sh script
--tags TAG1,TAG2	(Technology Preview) Use this option with a comma-separated list of tags from the config-download playbook to run the deployment with a specific set of config-download tasks.

Parameter	Description
--skip-tags TAG1,TAG2	**(Technology Preview)** Use this option with a comma-separated list of tags that you want to skip from the config-download playbook.

Run the following command to view a full list of options:

```
(undercloud) $ openstack help overcloud deploy
```

Some command line parameters are outdated or deprecated in favor of using heat template parameters, which you include in the **parameter_defaults** section in an environment file. The following table maps deprecated parameters to their heat template equivalents.

Table 7.9. Mapping deprecated CLI parameters to heat template parameters

Parameter	Description	Heat template parameter
--validation-errors-fatal	The overcloud creation process performs a set of pre-deployment checks. This option exits if any fatal errors occur from the pre-deployment checks. It is advisable to use this option because any errors can cause your deployment to fail.	No parameter mapping
--disable-validations	Disable the pre-deployment validations entirely. These validations were built-in pre-deployment validations, which have been replaced with external validations from the **openstack-tripleo-validations** package.	No parameter mapping
--config-download	Run deployment using the **config-download** mechanism. This is now the default and this CLI options may be removed in the future.	No parameter mapping
--rhel-reg	Use this option to register overcloud nodes to the Customer Portal or Satellite 6.	**RhsmVars**

Parameter	Description	Heat template parameter
--reg-method	Use this option to define the registration method that you want to use for the overcloud nodes. **satellite** for Red Hat Satellite 6 or Red Hat Satellite 5, **portal** for Customer Portal.	**RhsmVars**
--reg-org [REG_ORG]	The organization that you want to use for registration.	**RhsmVars**
--reg-force	Use this option to register the system even if it is already registered.	**RhsmVars**
--reg-sat-url [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. The overcloud creation process uses this URL to determine whether the server is a Red Hat Satellite 5 or Red Hat Satellite 6 server. If the server is a Red Hat Satellite 6 server, the overcloud obtains the **katello-ca-consumer-latest.noarch.rpm** file, registers with **subscription-manager**, and installs **katello-agent**. If the server is a Red Hat Satellite 5 server, the overcloud obtains the **RHN-ORG-TRUSTED-SSL-CERT** file and registers with **rhnreg_ks**.	**RhsmVars**
--reg-activation-key [REG_ACTIVATION_KEY]	Use this option to define the activation key that you want to use for registration.	**RhsmVars**

These parameters are scheduled for removal in a future version of Red Hat OpenStack Platform.

7.3.8. Contents of the default overcloud directory

In RHOSP 17, you can find all the configuration files in a single directory. The name of the directory is a combination of the openstack command used and the name of the stack. The directories have default locations but you can change the default locations by using the **--working-dir** option. You can use this option with any **tripleoclient** command that reads or creates files used with the deployment.

Default location	Command
$HOME/tripleo-deploy/undercloud	**undercloud install**, which is based on**tripleo deploy**
$HOME/tripleo-deploy/<stack>	**tripleo deploy**, <stack> is **standalone** by default
$HOME/overcloud-deploy/<stack>	**overcloud deploy**, <stack> is **overcloud** by default

The following table details the files and directories contained in the ~/**overcloud-deploy**/**overcloud** directory.

Table 7.10. Overcloud directory contents

Directory	Description

cli-*`	Directories used by **ansible-runner** for CLI ansible-based workflows:
	
	   cli-config-download
	   cli-enable-ssh-admin
	   cli-grant-local-access
	   cli-undercloud-get-horizon-url

config-download	The **config-download** directory. In earlier Red Hat OpenStack Platform (RHOSP) releases this directory was named ~/**config-download** or /**var**/**lib**/**mistral**/**<stack>**.
environment	Contains the saved stack environment generated with the **openstack stack environment show <stack>** command.
heat-launcher	The ephemeral Heat working directory containing the ephemeral Heat configuration and database backups.
outputs	Contains saved stack outputs generated with the **openstack stack output show <stack> <output>** command.
<stack>-deployment_status.yaml	Contains the saved stack status. Where **overcloud** is the name of the stack, this file is named **overcloud-deployment_status.yaml**.
<stack>-export.yaml	Contains stack export information, generated with the **openstack overcloud export --stack <stack>** command. Where **overcloud** is the name of the stack, this file is named **overcloud-export.yaml**.
<stack>-install-*.tar.bzip2	A tarball of the working directory, for example, **overcloud-install-20220823213643.tar.bzip2**.

Directory	Description
<stack>-passwords.yaml	Contains the stack passwords. Where **overcloud** is the name of the stack, this file is named **overcloud-passwords.yaml**.
<stack>rc	Credential file required to use the overcloud APIs, for example, **overcloudrc**.
tempest-deployer-input.conf	Contains the Tempest configuration.
tripleo-ansible-inventory.yaml	Ansible inventory for the overcloud.
tripleo-heat-templates	Contains a copy of rendered jinja2 templates. You can find the source templates in **/usr/share/openstack-tripleo-heat-templates** or you can specify the templates with the**--templates** options on the CLI.
tripleo-overcloud-baremetal-deployment.yaml	Baremetal deployment input for provisioning overcloud nodes.
tripleo-overcloud-network-data.yaml	Network deployment input for provisioning overcloud networks.
tripleo-overcloud-roles-data.yaml	Roles data which is specified with the **-r** option on the CLI.
tripleo-overcloud-virtual-ips.yaml	VIP deployment input for provisioning overcloud network VIPs.

7.3.9. Validating your overcloud deployment

Validate your deployed overcloud.

Prerequisites

- You have deployed your overcloud.

Procedure

1. Source the **stackrc** credentials file:

   ```
   $ source ~/stackrc
   ```

2. Validate your overcloud deployment:

   ```
   $ validation run \
     --group post-deployment \
     [--inventory <inventory_file>]
   ```

   - Replace **<inventory_file>** with the name of your ansible inventory file. By default, the dynamic inventory is called **tripleo-ansible-inventory**.

NOTE

When you run a validation, the **Reasons** column in the output is limited to 79 characters. To view the validation result in full, view the validation log files.

3. Review the results of the validation report:

   ```
   $ validation history get --full <UUID>
   ```

   - Replace **<UUID>** with the UUID of the validation run.

   - Optional: Use the **--full** option to view detailed output from the validation run.

IMPORTANT

A **FAILED** validation does not prevent you from deploying or running Red Hat OpenStack Platform. However, a **FAILED** validation can indicate a potential issue with a production environment.

7.3.10. Accessing the overcloud

Director generates a credential file containing the credentials necessary to operate the overcloud from the undercloud. Director saves this file, **overcloudrc**, in the home directory of the **stack** user.

Procedure

1. Source the **overcloudrc** file:

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

   The command prompt changes to indicate that you are accessing the overcloud:

   ```
   (overcloud)$
   ```

2. To return to interacting with the undercloud, source the **stackrc** file:

   ```
   (overcloud)$ source ~/stackrc
   (undercloud)$
   ```

   The command prompt changes to indicate that you are accessing the undercloud:

   ```
   (undercloud)$
   ```

7.4. CONFIGURING A BASIC OVERCLOUD WITH PRE-PROVISIONED NODES

This chapter contains basic configuration procedures that you can use to configure a Red Hat OpenStack Platform (RHOSP) environment with pre-provisioned nodes. This scenario differs from the standard overcloud creation scenarios in several ways:

- You can provision nodes with an external tool and let the director control the overcloud configuration only.

- You can use nodes without relying on the director provisioning methods. This is useful if you want to create an overcloud without power management control, or use networks with DHCP/PXE boot restrictions.

- The director does not use OpenStack Compute (nova), OpenStack Bare Metal (ironic), or OpenStack Image (glance) to manage nodes.

- Pre-provisioned nodes can use a custom partitioning layout that does not rely on the QCOW2 overcloud-full image.

This scenario includes only basic configuration with no custom features.

> **IMPORTANT**
>
> You cannot combine pre-provisioned nodes with director-provisioned nodes.

7.4.1. Pre-provisioned node requirements

Before you begin deploying an overcloud with pre-provisioned nodes, ensure that the following configuration is present in your environment:

- The director node that you created in Chapter 4, *Installing director on the undercloud* .

- A set of bare metal machines for your nodes. The number of nodes required depends on the type of overcloud you intend to create. These machines must comply with the requirements set for each node type. These nodes require Red Hat Enterprise Linux 9.2 installed as the host operating system. Red Hat recommends using the latest version available.

- One network connection for managing the pre-provisioned nodes. This scenario requires uninterrupted SSH access to the nodes for orchestration agent configuration.

- One network connection for the Control Plane network. There are two main scenarios for this network:

  - Using the Provisioning Network as the Control Plane, which is the default scenario. This network is usually a layer-3 (L3) routable network connection from the pre-provisioned nodes to director. The examples for this scenario use following IP address assignments:

    Table 7.11. Provisioning Network IP assignments

Node name	IP address
Director	192.168.24.1
Controller 0	192.168.24.2
Compute 0	192.168.24.3

  - Using a separate network. In situations where the director's Provisioning network is a private non-routable network, you can define IP addresses for nodes from any subnet and communicate with director over the Public API endpoint. For more information about the requirements for this scenario, see Section 7.4.6, "Using a separate network for pre-provisioned nodes".

- All other network types in this example also use the Control Plane network for OpenStack services. However, you can create additional networks for other network traffic types.

- If any nodes use Pacemaker resources, the service user **hacluster** and the service group **haclient** must have a UID/GID of **189**. This is due to CVE-2018-16877. If you installed Pacemaker together with the operating system, the installation creates these IDs automatically. If the ID values are set incorrectly, follow the steps in the article OpenStack minor update / fast-forward upgrade can fail on the controller nodes at pacemaker step with "Could not evaluate: backup_cib" to change the ID values.

- To prevent some services from binding to an incorrect IP address and causing deployment failures, make sure that the **/etc/hosts** file does not include the **node-name=127.0.0.1** mapping.

7.4.2. Creating a user on pre-provisioned nodes

When you configure an overcloud with pre-provisioned nodes, director requires SSH access to the overcloud nodes. On the pre-provisioned nodes, you must create a user with SSH key authentication and configure passwordless sudo access for that user. After you create a user on pre-provisioned nodes, you can use the **--overcloud-ssh-user** and **--overcloud-ssh-key** options with the **openstack overcloud deploy** command to create an overcloud with pre-provisioned nodes.

By default, the values for the overcloud SSH user and overcloud SSH key are the **stack** user and **~/.ssh/id_rsa**. To create the **stack** user, complete the following steps.

Procedure

1. On each overcloud node, create the **stack** user and set a password. For example, run the following commands on the Controller node:

   ```
   [root@controller-0 ~]# useradd stack
   [root@controller-0 ~]# passwd stack  # specify a password
   ```

2. Disable password requirements for this user when using **sudo**:

   ```
   [root@controller-0 ~]# echo "stack ALL=(root) NOPASSWD:ALL" | tee -a
   /etc/sudoers.d/stack
   [root@controller-0 ~]# chmod 0440 /etc/sudoers.d/stack
   ```

3. After you create and configure the **stack** user on all pre-provisioned nodes, copy the **stack** user's public SSH key from the director node to each overcloud node. For example, to copy the director's public SSH key to the Controller node, run the following command:

   ```
   [stack@director ~]$ ssh-copy-id stack@192.168.24.2
   ```

IMPORTANT

To copy your SSH keys, you might have to temporarily set **PasswordAuthentication Yes** in the SSH configuration of each overcloud node. After you copy the SSH keys, set **PasswordAuthentication No** and use the SSH keys to authenticate in the future.

7.4.3. Registering the operating system for pre-provisioned nodes

Each node requires access to a Red Hat subscription. Complete the following steps on each node to register your nodes with the Red Hat Content Delivery Network. Execute the commands as the **root** user or as a user with **sudo** privileges.

> **IMPORTANT**
>
> Enable only the repositories listed. Additional repositories can cause package and software conflicts. Do not enable any additional repositories.

Procedure

1. Run the registration command and enter your Customer Portal user name and password when prompted:

   ```
   [root@controller-0 ~]# sudo subscription-manager register
   ```

2. Find the entitlement pool for Red Hat OpenStack Platform (RHOSP) 17.1:

   ```
   [root@controller-0 ~]# sudo subscription-manager list --available --all --matches="Red Hat OpenStack"
   ```

3. Use the pool ID located in the previous step to attach the RHOSP 17.1 entitlements:

   ```
   [root@controller-0 ~]# sudo subscription-manager attach --pool=pool_id
   ```

4. Disable all default repositories:

   ```
   [root@controller-0 ~]# sudo subscription-manager repos --disable=*
   ```

5. Enable the required Red Hat Enterprise Linux (RHEL) repositories:

   ```
   [root@controller-0 ~]# sudo subscription-manager repos \
     --enable=rhel-9-for-x86_64-baseos-eus-rpms \
     --enable=rhel-9-for-x86_64-appstream-eus-rpms \
     --enable=rhel-9-for-x86_64-highavailability-eus-rpms \
     --enable=openstack-17.1-for-rhel-9-x86_64-rpms \
     --enable=fast-datapath-for-rhel-9-x86_64-rpms
   ```

6. If the overcloud uses Red Hat Ceph Storage nodes, enable the relevant Red Hat Ceph Storage repositories:

   ```
   [root@cephstorage-0 ~]# sudo subscription-manager repos \
     --enable=rhel-9-for-x86_64-baseos-rpms \
     --enable=rhel-9-for-x86_64-appstream-rpms \
     --enable=openstack-deployment-tools-for-rhel-9-x86_64-rpms
   ```

7. Lock the RHEL version on all overcloud nodes except Red Hat Ceph Storage nodes:

   ```
   [root@controller-0 ~]# subscription-manager release --set=9.2
   ```

8. Update your system to ensure you have the latest base system packages:

```
[root@controller-0 ~]# sudo dnf update -y
[root@controller-0 ~]# sudo reboot
```

The node is now ready to use for your overcloud.

7.4.4. Configuring SSL/TLS access to director

If the director uses SSL/TLS, the pre-provisioned nodes require the certificate authority file used to sign the director's SSL/TLS certificates. If you use your own certificate authority, perform the following actions on each overcloud node.

Procedure

1. Copy the certificate authority file to the **/etc/pki/ca-trust/source/anchors/** directory on each pre-provisioned node.

2. Run the following command on each overcloud node:

   ```
   [root@controller-0 ~]#  sudo update-ca-trust extract
   ```

These steps ensure that the overcloud nodes can access the director's Public API over SSL/TLS.

7.4.5. Configuring networking for the control plane

The pre-provisioned overcloud nodes obtain metadata from director using standard HTTP requests. This means all overcloud nodes require L3 access to either:

- The director Control Plane network, which is the subnet that you define with the **network_cidr** parameter in your **undercloud.conf** file. The overcloud nodes require either direct access to this subnet or routable access to the subnet.

- The director Public API endpoint, that you specify with the **undercloud_public_host** parameter in your **undercloud.conf** file. This option is available if you do not have an L3 route to the Control Plane or if you want to use SSL/TLS communication. For more information about configuring your overcloud nodes to use the Public API endpoint, see Section 7.4.6, "Using a separate network for pre-provisioned nodes".

Director uses the Control Plane network to manage and configure a standard overcloud. For an overcloud with pre-provisioned nodes, your network configuration might require some modification to accommodate communication between the director and the pre-provisioned nodes.

Using network isolation

You can use network isolation to group services to use specific networks, including the Control Plane. You can also define specific IP addresses for nodes on the Control Plane.
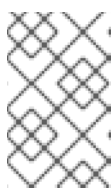
> **NOTE**
>
> If you use network isolation, ensure that your NIC templates do not include the NIC used for undercloud access. These templates can reconfigure the NIC, which introduces connectivity and configuration problems during deployment.

Assigning IP addresses

If you do not use network isolation, you can use a single Control Plane network to manage all services. This requires manual configuration of the Control Plane NIC on each node to use an IP address within the Control Plane network range. If you are using the director Provisioning network as the Control Plane, ensure that the overcloud IP addresses that you choose are outside of the DHCP ranges for both provisioning (**dhcp_start** and **dhcp_end**) and introspection (**inspection_iprange**).

During standard overcloud creation, director creates OpenStack Networking (neutron) ports and automatically assigns IP addresses to the overcloud nodes on the Provisioning / Control Plane network. However, this can cause director to assign different IP addresses to the ones that you configure manually for each node. In this situation, use a predictable IP address strategy to force director to use the pre-provisioned IP assignments on the Control Plane.

If you are using network isolation, create a custom environment file, **deployed-ports.yaml**, to implement a predictable IP strategy. The following example custom environment file, **deployed-ports.yaml**, passes a set of resource registry mappings and parameters to director, and defines the IP assignments of the pre-provisioned nodes. The **NodePortMap**, **ControlPlaneVipData**, and **VipPortMap** parameters define the IP addresses and subnet CIDRs that correspond to each overcloud node.

```
resource_registry:
  # Deployed Virtual IP port resources
  OS::TripleO::Network::Ports::ExternalVipPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/deployed_vip_external.yaml
  OS::TripleO::Network::Ports::InternalApiVipPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/deployed_vip_internal_api.yaml
  OS::TripleO::Network::Ports::StorageVipPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/deployed_vip_storage.yaml
  OS::TripleO::Network::Ports::StorageMgmtVipPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/deployed_vip_storage_mgmt.yaml

  # Deployed ControlPlane port resource
  OS::TripleO::DeployedServer::ControlPlanePort: /usr/share/openstack-tripleo-heat-
templates/deployed-server/deployed-neutron-port.yaml

  # Controller role port resources
  OS::TripleO::Controller::Ports::ExternalPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/deployed_external.yaml
  OS::TripleO::Controller::Ports::InternalApiPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/deployed_internal_api.yaml
  OS::TripleO::Controller::Ports::StorageMgmtPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/deployed_storage_mgmt.yaml
  OS::TripleO::Controller::Ports::StoragePort: /usr/share/openstack-tripleo-heat-
templates/network/ports/deployed_storage.yaml
  OS::TripleO::Controller::Ports::TenantPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/deployed_tenant.yaml

  # Compute role port resources
  OS::TripleO::Compute::Ports::InternalApiPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/deployed_internal_api.yaml
  OS::TripleO::Compute::Ports::StoragePort: /usr/share/openstack-tripleo-heat-
templates/network/ports/deployed_storage.yaml
  OS::TripleO::Compute::Ports::TenantPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/deployed_tenant.yaml

  # CephStorage role port resources
  OS::TripleO::CephStorage::Ports::StorageMgmtPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/deployed_storage_mgmt.yaml
```

```
  OS::TripleO::CephStorage::Ports::StoragePort: /usr/share/openstack-tripleo-heat-
templates/network/ports/deployed_storage.yaml

parameter_defaults:
  NodePortMap: 1
    # Controller node parameters
    controller-00-rack01: 2
      ctlplane: 3
        ip_address: 192.168.24.201
        ip_address_uri: 192.168.24.201
        ip_subnet: 192.168.24.0/24
      external:
        ip_address: 10.0.0.201
        ip_address_uri: 10.0.0.201
        ip_subnet: 10.0.0.10/24
      internal_api:
        ip_address: 172.16.2.201
        ip_address_uri: 172.16.2.201
        ip_subnet: 172.16.2.10/24
      management:
        ip_address: 192.168.1.201
        ip_address_uri: 192.168.1.201
        ip_subnet: 192.168.1.10/24
      storage:
        ip_address: 172.16.1.201
        ip_address_uri: 172.16.1.201
        ip_subnet: 172.16.1.10/24
      storage_mgmt:
        ip_address: 172.16.3.201
        ip_address_uri: 172.16.3.201
        ip_subnet: 172.16.3.10/24
      tenant:
        ip_address: 172.16.0.201
        ip_address_uri: 172.16.0.201
        ip_subnet: 172.16.0.10/24
    ...

    # Compute node parameters
    compute-00-rack01:
      ctlplane:
        ip_address: 192.168.24.11
        ip_address_uri: 192.168.24.11
        ip_subnet: 192.168.24.0/24
      internal_api:
        ip_address: 172.16.2.11
        ip_address_uri: 172.16.2.11
        ip_subnet: 172.16.2.10/24
      storage:
        ip_address: 172.16.1.11
        ip_address_uri: 172.16.1.11
        ip_subnet: 172.16.1.10/24
      tenant:
        ip_address: 172.16.0.11
        ip_address_uri: 172.16.0.11
        ip_subnet: 172.16.0.10/24
    ...
```

```
  # Ceph node parameters
  ceph-00-rack01:
    ctlplane:
      ip_address: 192.168.24.101
      ip_address_uri: 192.168.24.101
      ip_subnet: 192.168.24.0/24
    storage:
      ip_address: 172.16.1.101
      ip_address_uri: 172.16.1.101
      ip_subnet: 172.16.1.10/24
    storage_mgmt:
      ip_address: 172.16.3.101
      ip_address_uri: 172.16.3.101
      ip_subnet: 172.16.3.10/24

  ...

# Virtual IP address parameters
ControlPlaneVipData:
  fixed_ips:
  - ip_address: 192.168.24.5
  name: control_virtual_ip
  network:
    tags: [192.168.24.0/24]
    subnets:
    - ip_version: 4
VipPortMap
  external:
    ip_address: 10.0.0.100
    ip_address_uri: 10.0.0.100
    ip_subnet: 10.0.0.100/24
  internal_api:
    ip_address: 172.16.2.100
    ip_address_uri: 172.16.2.100
    ip_subnet: 172.16.2.100/24
  storage:
    ip_address: 172.16.1.100
    ip_address_uri: 172.16.1.100
    ip_subnet: 172.16.1.100/24
  storage_mgmt:
    ip_address: 172.16.3.100
    ip_address_uri: 172.16.3.100
    ip_subnet: 172.16.3.100/24

RedisVirtualFixedIPs:
  - ip_address: 192.168.24.6
      use_neutron: false
```

1 The **NodePortMap** mappings define the names of the node assignments.

2 The short host name for the node, which follows the format **<node_hostname>**.

3 The network definitions and IP assignments for the node. Networks include **ctlplane**, **external**, **internal_api**, **management**, **storage**, **storage_mgmt**, and **tenant**. The IP assignments include the **ip_address**, the **ip_address_uri**, and the **ip_subnet**:

- IPv4: **ip_address** and **ip_address_uri** should be set to the same value.

- IPv6:

  - **ip_address**: Set to the IPv6 address without brackets.

  - **ip_address_uri**: Set to the IPv6 address in square brackets, for example,
    [2001:0db8:85a3:0000:0000:8a2e:0370:7334].

7.4.6. Using a separate network for pre-provisioned nodes

By default, director uses the Provisioning network as the overcloud Control Plane. However, if this
network is isolated and non-routable, nodes cannot communicate with the director Internal API during
configuration. In this situation, you might need to define a separate network for the nodes and configure
them to communicate with the director over the Public API.

There are several requirements for this scenario:

- The overcloud nodes must accommodate the basic network configuration from Section 7.4.5,
  "Configuring networking for the control plane".

- You must enable SSL/TLS on the director for Public API endpoint usage. For more information,
  see Enabling SSL/TLS on overcloud public endpoints .

- You must define an accessible fully qualified domain name (FQDN) for director. This FQDN
  must resolve to a routable IP address for the director. Use the **undercloud_public_host**
  parameter in the **undercloud.conf** file to set this FQDN.

The examples in this section use IP address assignments that differ from the main scenario:

Table 7.12. Provisioning network IP assignments

Node Name	IP address or FQDN
Director (Internal API)	192.168.24.1 (Provisioning Network and Control Plane)
Director (Public API)	10.1.1.1 / director.example.com
Overcloud Virtual IP	192.168.100.1
Controller 0	192.168.100.2
Compute 0	192.168.100.3

The following sections provide additional configuration for situations that require a separate network for
overcloud nodes.

IP address assignments

The method for IP assignments is similar to Section 7.4.5, "Configuring networking for the control
plane". However, since the Control Plane may not be routable from the deployed servers, you can use
the **NodePortMap**, **ControlPlaneVipData**, and **VipPortMap** parameters to assign IP addresses from

your chosen overcloud node subnet, including the virtual IP address to access the Control Plane. The following example is a modified version of the **deployed-ports.yaml** custom environment file from Section 7.4.5, "Configuring networking for the control plane" that accommodates this network architecture:

```
parameter_defaults:
  NodePortMap:
    controller-00-rack01
      ctlplane
        ip_address: 192.168.100.2
        ip_address_uri: 192.168.100.2
        ip_subnet: 192.168.100.0/24
...
    compute-00-rack01:
      ctlplane
        ip_address: 192.168.100.3
        ip_address_uri: 192.168.100.3
        ip_subnet: 192.168.100.0/24
...
  ControlPlaneVipData:
    fixed_ips:
    - ip_address: 192.168.100.1
    name: control_virtual_ip
    network:
      tags: [192.168.100.0/24]
    subnets:
    - ip_version: 4
  VipPortMap:
    external:
      ip_address: 10.0.0.100
      ip_address_uri: 10.0.0.100
      ip_subnet: 10.0.0.100/24
....
  RedisVirtualFixedIPs: **(1)**
    - ip_address: 192.168.100.10
    use_neutron: false
```

(1) The **RedisVipPort** resource is mapped to **network/ports/noop.yaml**. This mapping is necessary because the default Redis VIP address comes from the Control Plane. In this situation, use a **noop** to disable this Control Plane mapping.

7.4.7. Mapping pre-provisioned node hostnames

When you configure pre-provisioned nodes, you must map heat-based hostnames to their actual hostnames so that **ansible-playbook** can reach a resolvable host. Use the **HostnameMap** to map these values.

Procedure

1. Create an environment file, for example **hostname-map.yaml**, and include the **HostnameMap** parameter and the hostname mappings. Use the following syntax:

```
parameter_defaults:
  HostnameMap:
```

```
[HEAT HOSTNAME]: [ACTUAL HOSTNAME]
[HEAT HOSTNAME]: [ACTUAL HOSTNAME]
```

The **[HEAT HOSTNAME]** usually conforms to the following convention: **[STACK NAME]-[ROLE]-[INDEX]**:

```
parameter_defaults:
  HostnameMap:
    overcloud-controller-0: controller-00-rack01
    overcloud-controller-1: controller-01-rack02
    overcloud-controller-2: controller-02-rack03
    overcloud-novacompute-0: compute-00-rack01
    overcloud-novacompute-1: compute-01-rack01
    overcloud-novacompute-2: compute-02-rack01
```

2. Save the **hostname-map.yaml** file.

7.4.8. Configuring Ceph Storage for pre-provisioned nodes

Complete the following steps on the undercloud host to configure Ceph for nodes that are already deployed.

Procedure

1. On the undercloud host, create an environment variable, **OVERCLOUD_HOSTS**, and set the variable to a space-separated list of IP addresses of the overcloud hosts that you want to use as Ceph clients:

   ```
   export OVERCLOUD_HOSTS="192.168.1.8 192.168.1.42"
   ```

2. The default overcloud plan name is **overcloud**. If you use a different name, create an environment variable **OVERCLOUD_PLAN** to store your custom name:

   ```
   export OVERCLOUD_PLAN="<custom-stack-name>"
   ```

   - Replace **<custom-stack-name>** with the name of your stack.

3. Run the **enable-ssh-admin.sh** script to configure a user on the overcloud nodes that Ansible can use to configure Ceph clients:

   ```
   bash /usr/share/openstack-tripleo-heat-templates/deployed-server/scripts/enable-ssh-admin.sh
   ```

When you run the **openstack overcloud deploy** command, Ansible configures the hosts that you define in the **OVERCLOUD_HOSTS** variable as Ceph clients.

7.4.9. Creating the overcloud with pre-provisioned nodes

The overcloud deployment uses the standard CLI methods. For pre-provisioned nodes, the deployment command requires some additional options and environment files from the core heat template collection:

- **--disable-validations** - Use this option to disable basic CLI validations for services not used with pre-provisioned infrastructure. If you do not disable these validations, the deployment fails.

- **environments/deployed-server-environment.yaml** – Include this environment file to create and configure the pre-provisioned infrastructure. This environment file substitutes the **OS::Nova::Server** resources with **OS::Heat::DeployedServer** resources.

The following command is an example overcloud deployment command with the environment files specific to the pre-provisioned architecture:

```
$ source ~/stackrc
(undercloud)$ openstack overcloud deploy \
  --disable-validations \
  -e /usr/share/openstack-tripleo-heat-templates/environments/deployed-server-environment.yaml \
  -e /home/stack/templates/deployed-ports.yaml \
  -e /home/stack/templates/hostname-map.yaml \
  --overcloud-ssh-user stack \
  --overcloud-ssh-key ~/.ssh/id_rsa \
  <OTHER OPTIONS>
```

The **--overcloud-ssh-user** and **--overcloud-ssh-key** options are used to SSH into each overcloud node during the configuration stage, create an initial **tripleo-admin** user, and inject an SSH key into **/home/tripleo-admin/.ssh/authorized_keys**. To inject the SSH key, specify the credentials for the initial SSH connection with **--overcloud-ssh-user** and **--overcloud-ssh-key** (defaults to **~/.ssh/id_rsa**). To limit exposure to the private key that you specify with the **--overcloud-ssh-key** option, director never passes this key to any API service, such as heat, and only the director **openstack overcloud deploy** command uses this key to enable access for the **tripleo-admin** user.

7.4.10. Accessing the overcloud

Director generates a credential file containing the credentials necessary to operate the overcloud from the undercloud. Director saves this file, **overcloudrc**, in the home directory of the **stack** user.

Procedure

1. Source the **overcloudrc** file:

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

   The command prompt changes to indicate that you are accessing the overcloud:

   ```
   (overcloud)$
   ```

2. To return to interacting with the undercloud, source the **stackrc** file:

   ```
   (overcloud)$ source ~/stackrc
   (undercloud)$
   ```

   The command prompt changes to indicate that you are accessing the undercloud:

   ```
   (undercloud)$
   ```

7.4.11. Scaling pre-provisioned nodes

The process for scaling pre-provisioned nodes is similar to the standard scaling procedures in Scaling overcloud nodes. However, the process to add new pre-provisioned nodes differs because pre-

provisioned nodes do not use the standard registration and management process from the Bare Metal Provisioning service (ironic) and the Compute service (nova).

7.4.11.1. Scaling up pre-provisioned nodes

When scaling up the overcloud with pre-provisioned nodes, you must configure the orchestration agent on each node to correspond to the director node count.

Procedure

1. Prepare the new pre-provisioned nodes. For more information, see Pre-provisioned node requirements.

2. Scale up the nodes. For more information, see Scaling overcloud nodes .

7.4.11.2. Scaling down pre-provisioned nodes

When scaling down an overcloud that has pre-provisioned nodes, follow the scale down instructions in Scaling overcloud nodes .

In scale-down operations, you can use host names for both Red Hat OpenStack Platform (RHOSP) provisioned or pre-provisioned nodes. You can also use the UUID for RHOSP provisioned nodes. However, there is no UUID for pre-provisoned nodes, so you always use the host name.

Procedure

1. Retrieve the names of the nodes that you want to remove:

   ```
   $ openstack stack resource list overcloud -n5 --filter
   type=OS::TripleO::ComputeDeployedServerServer
   ```

2. Delete the nodes:

   ```
   $ openstack overcloud node delete --stack <overcloud> <node> [... <node>]
   ```

   - Replace **<overcloud>** with the name or UUID of the overcloud stack.

   - Replace **<node>** with the host names of the nodes that you want to remove, retrieved from the **stack_name** column returned in step 1.

3. Ensure that the node is deleted:

   ```
   $ openstack stack list
   ```

   The status of the **overcloud** stack shows **UPDATE_COMPLETE** when the delete operation is complete.

4. Power off the removed nodes. In a standard deployment, the bare-metal services on director power off the removed nodes. With pre-provisioned nodes, you must either manually shut down the removed nodes or use the power management control for each physical system. If you do not power off the nodes after removing them from the stack, they might remain operational and reconnect as part of the overcloud environment.

5. Re-provision the removed nodes to a base operating system configuration so that they do not unintentionally join the overcloud in the future.

NOTE

Do not attempt to reuse nodes previously removed from the overcloud without first reprovisioning them with a fresh base operating system. The scale down process only removes the node from the overcloud stack and does not uninstall any packages.

7.4.12. Removing a pre-provisioned overcloud

To remove an entire overcloud that uses pre-provisioned nodes, see Section 9.7, "Removing an overcloud stack" for the standard overcloud removal procedure. After you remove the overcloud, power off all nodes and reprovision them to a base operating system configuration.

NOTE

Do not attempt to reuse nodes previously removed from the overcloud without first reprovisioning them with a fresh base operating system. The removal process only deletes the overcloud stack and does not uninstall any packages.

CHAPTER 8. PERFORMING OVERCLOUD POST-INSTALLATION TASKS

This chapter contains information about tasks to perform immediately after you create your overcloud. These tasks ensure your overcloud is ready to use.

8.1. CHECKING OVERCLOUD DEPLOYMENT STATUS

To check the deployment status of the overcloud, use the **openstack overcloud status** command. This command returns the result of all deployment steps.

Procedure

1. Source the **stackrc** file:

   ```
   $ source ~/stackrc
   ```

2. Run the deployment status command:

   ```
   $ openstack overcloud status
   ```

   The output of this command displays the status of the overcloud:

   ```
   +-----------+---------------------+---------------------+-------------------+
   | Plan Name |       Created       |       Updated       | Deployment Status |
   +-----------+---------------------+---------------------+-------------------+
   | overcloud | 2018-05-03 21:24:50 | 2018-05-03 21:27:59 |   DEPLOY_SUCCESS  |
   +-----------+---------------------+---------------------+-------------------+
   ```

   If your overcloud uses a different name, use the **--stack** argument to select an overcloud with a different name:

   ```
   $ openstack overcloud status --stack <overcloud_name>
   ```

   - Replace **<overcloud_name>** with the name of your overcloud.

8.2. CREATING BASIC OVERCLOUD FLAVORS

Validation steps in this guide assume that your installation contains flavors. If you have not already created at least one flavor, complete the following steps to create a basic set of default flavors that have a range of storage and processing capabilities:

Procedure

1. Source the **overcloudrc** file:

   ```
   $ source ~/overcloudrc
   ```

2. Run the **openstack flavor create** command to create a flavor. Use the following options to specify the hardware requirements for each flavor:

   --disk

Defines the hard disk space for a virtual machine volume.

--ram

Defines the RAM required for a virtual machine.

--vcpus

Defines the quantity of virtual CPUs for a virtual machine.

3. The following example creates the default overcloud flavors:

```
$ openstack flavor create m1.tiny --ram 512 --disk 0 --vcpus 1
$ openstack flavor create m1.smaller --ram 1024 --disk 0 --vcpus 1
$ openstack flavor create m1.small --ram 2048 --disk 10 --vcpus 1
$ openstack flavor create m1.medium --ram 3072 --disk 10 --vcpus 2
$ openstack flavor create m1.large --ram 8192 --disk 10 --vcpus 4
$ openstack flavor create m1.xlarge --ram 8192 --disk 10 --vcpus 8
```

> **NOTE**
>
> Use **$ openstack flavor create --help** to learn more about the **openstack flavor create** command.

8.3. CREATING A DEFAULT TENANT NETWORK

The overcloud requires a default Tenant network so that virtual machines can communicate internally.

Procedure

1. Source the **overcloudrc** file:

   ```
   $ source ~/overcloudrc
   ```

2. Create the default Tenant network:

   ```
   (overcloud) $ openstack network create default
   ```

3. Create a subnet on the network:

   ```
   (overcloud) $ openstack subnet create default --network default --gateway 172.20.1.1 --subnet-range 172.20.0.0/16
   ```

4. Confirm the created network:

   ```
   (overcloud) $ openstack network list
   +----------------------+------------+-------------------------------------+
   | id                   | name       | subnets                             |
   +----------------------+------------+-------------------------------------+
   | 95fadaa1-5dda-4777... | default    | 7e060813-35c5-462c-a56a-1c6f8f4f332f |
   +----------------------+------------+-------------------------------------+
   ```

These commands create a basic Networking service (neutron) network named **default**. The overcloud automatically assigns IP addresses from this network to virtual machines using an internal DHCP mechanism.

8.4. CREATING A DEFAULT FLOATING IP NETWORK

To access your virtual machines from outside of the overcloud, you must configure an external network that provides floating IP addresses to your virtual machines.

This procedure contains two examples. Use the example that best suits your environment:

- Native VLAN (flat network)

- Non-Native VLAN (VLAN network)

Both of these examples involve creating a network with the name **public**. The overcloud requires this specific name for the default floating IP pool. This name is also important for the validation tests in Section 8.7, "Validating the overcloud".

By default, Openstack Networking (neutron) maps a physical network name called **datacentre** to the **br-ex** bridge on your host nodes. You connect the **public** overcloud network to the physical **datacentre** and this provides a gateway through the **br-ex** bridge.

Prerequisites

- A dedicated interface or native VLAN for the floating IP network.

Procedure

1. Source the **overcloudrc** file:

   ```
   $ source ~/overcloudrc
   ```

2. Create the **public** network:

   - Create a **flat** network for a native VLAN connection:

     ```
     (overcloud) $ openstack network create public --external --provider-network-type flat --provider-physical-network datacentre
     ```

   - Create a **vlan** network for non-native VLAN connections:

     ```
     (overcloud) $ openstack network create public --external --provider-network-type vlan --provider-physical-network datacentre --provider-segment 201
     ```

     Use the **--provider-segment** option to define the VLAN that you want to use. In this example, the VLAN is **201**.

3. Create a subnet with an allocation pool for floating IP addresses. In this example, the IP range is **10.1.1.51** to **10.1.1.250**:

   ```
   (overcloud) $ openstack subnet create public --network public --dhcp --allocation-pool start=10.1.1.51,end=10.1.1.250 --gateway 10.1.1.1 --subnet-range 10.1.1.0/24
   ```

   Ensure that this range does not conflict with other IP addresses in your external network.

8.5. CREATING A DEFAULT PROVIDER NETWORK

A provider network is another type of external network connection that routes traffic from private tenant networks to external infrastructure network. The provider network is similar to a floating IP network but the provider network uses a logical router to connect private networks to the provider network.

This procedure contains two examples. Use the example that best suits your environment:

- Native VLAN (flat network)

- Non-Native VLAN (VLAN network)

By default, Openstack Networking (neutron) maps a physical network name called **datacentre** to the **br-ex** bridge on your host nodes. You connect the **public** overcloud network to the physical **datacentre** and this provides a gateway through the **br-ex** bridge.

Procedure

1. Source the **overcloudrc** file:

   ```
   $ source ~/overcloudrc
   ```

2. Create the **provider** network:

   - Create a **flat** network for a native VLAN connection:

     ```
     (overcloud) $ openstack network create provider --external --provider-network-type flat --provider-physical-network datacentre --share
     ```

   - Create a **vlan** network for non-native VLAN connections:

     ```
     (overcloud) $ openstack network create provider --external --provider-network-type vlan --provider-physical-network datacentre --provider-segment 201 --share
     ```

     - Use the **--provider-segment** option to define the VLAN that you want to use. In this example, the VLAN is **201**.

     - Use the **--share** option to create a shared network. Alternatively, specify a tenant instead of specifying **--share**, so that only the tenant has access to the new network.

     - Use the **--external** option to mark a provider network as external. Only the operator can create ports on an external network.

3. Add a subnet to the **provider** network to provide DHCP services:

   ```
   (overcloud) $ openstack subnet create provider-subnet --network  provider --dhcp --allocation-pool start=10.9.101.50,end=10.9.101.100 --gateway 10.9.101.254 --subnet-range 10.9.101.0/24
   ```

4. Create a router so that other networks can route traffic through the provider network:

   ```
   (overcloud) $ openstack router create external
   ```

5. Set the external gateway for the router to the **provider** network:

   ```
   (overcloud) $ openstack router set --external-gateway provider external
   ```

6. Attach other networks to this router. For example, run the following command to attach a subnet **subnet1** to the router:

```
(overcloud) $ openstack router add subnet external subnet1
```

This command adds **subnet1** to the routing table and allows traffic from virtual machines using **subnet1** to route to the provider network.

8.6. CREATING ADDITIONAL BRIDGE MAPPINGS

Floating IP networks can use any bridge, not just **br-ex**, provided that you map the additional bridge during deployment.

Procedure

1. To map a new bridge called **br-floating** to the **floating** physical network, include the **NeutronBridgeMappings** parameter in an environment file:

```
parameter_defaults:
  NeutronBridgeMappings: "datacentre:br-ex,floating:br-floating"
```

2. With this method, you can create separate external networks after creating the overcloud. For example, to create a floating IP network that maps to the **floating** physical network, run the following commands:

```
$ source ~/overcloudrc
(overcloud) $ openstack network create public --external --provider-physical-network floating
--provider-network-type vlan --provider-segment 105
(overcloud) $ openstack subnet create public --network public --dhcp --allocation-pool
start=10.1.2.51,end=10.1.2.250 --gateway 10.1.2.1 --subnet-range 10.1.2.0/24
```

8.7. VALIDATING THE OVERCLOUD

The overcloud uses the OpenStack Integration Test Suite (tempest) tool set to conduct a series of integration tests. This section contains information about preparations for running the integration tests. For full instructions about how to use the OpenStack Integration Test Suite, see the Validating your cloud with the Red Hat OpenStack Platform Integration Test Suite.

The Integration Test Suite requires a few post-installation steps to ensure successful tests.

Procedure

1. If you run this test from the undercloud, ensure that the undercloud host has access to the Internal API network on the overcloud. For example, add a temporary VLAN on the undercloud host to access the Internal API network (ID: 201) using the 172.16.0.201/24 address:

```
$ source ~/stackrc
(undercloud) $ sudo ovs-vsctl add-port br-ctlplane vlan201 tag=201 -- set interface vlan201
type=internal
(undercloud) $ sudo ip l set dev vlan201 up; sudo ip addr add 172.16.0.201/24 dev vlan201
```

2. Run the integration tests as described in the Validating your cloud with the Red Hat OpenStack Platform Integration Test Suite.

3. After completing the validation, remove any temporary connections to the overcloud Internal API. In this example, use the following commands to remove the previously created VLAN on the undercloud:

```
$ source ~/stackrc
(undercloud) $ sudo ovs-vsctl del-port vlan201
```

8.8. PROTECTING THE OVERCLOUD FROM REMOVAL

You can set a custom policy for the Orchestration service (heat) to protect your overcloud from being deleted.

To re-enable stack deletion, remove the **prevent-stack-delete.yaml** file from the **custom_env_files** parameter and run the **openstack undercloud install** command.

Procedure

1. Create an environment file named **prevent-stack-delete.yaml**.

2. Set the **HeatApiPolicies** parameter:

```
parameter_defaults:
  HeatApiPolicies:
    heat-deny-action:
      key: 'actions:action'
      value: 'rule:deny_everybody'
    heat-protect-overcloud:
      key: 'stacks:delete'
      value: 'rule:deny_everybody'
```

- The **heat-deny-action** is a default policy that you must include in your undercloud installation.

- Set the **heat-protect-overcloud** policy to **rule:deny_everybody** to prevent anyone from deleting any stacks in the overcloud.

  > **NOTE**
  >
  > Setting the overcloud protection to **rule:deny_everybody** means that you cannot perform any of the following functions:
  >
  > - Delete the overcloud.
  >
  > - Remove individual Compute or Storage nodes.
  >
  > - Replace Controller nodes.

3. Add the **prevent-stack-delete.yaml** environment file to the **custom_env_files** parameter in the **undercloud.conf** file:

```
custom_env_files = prevent-stack-delete.yaml
```

4. Run the undercloud installation command to refresh the configuration:

```
$ openstack undercloud install
```

CHAPTER 9. PERFORMING BASIC OVERCLOUD ADMINISTRATION TASKS

This chapter contains information about basic tasks you might need to perform during the lifecycle of your overcloud.

9.1. ACCESSING OVERCLOUD NODES THROUGH SSH

You can access each overcloud node through the SSH protocol.

- Each overcloud node contains a **tripleo-admin** user, formerly known as the **heat-admin** user.

- The **stack** user on the undercloud has key-based SSH access to the **tripleo-admin** user on each overcloud node.

- All overcloud nodes have a short hostname that the undercloud resolves to an IP address on the control plane network. Each short hostname uses a **.ctlplane** suffix. For example, the short name for **overcloud-controller-0** is **overcloud-controller-0.ctlplane**

Prerequisites

- A deployed overcloud with a working control plane network.

Procedure

1. Log in to the undercloud as the **stack** user.

2. Find the name of the node that you want to access:

   ```
   (undercloud)$ metalsmith list
   ```

3. Connect to the node as the **tripleo-admin** user:

   ```
   (undercloud)$ ssh tripleo-admin@overcloud-controller-0
   ```

9.2. MANAGING CONTAINERIZED SERVICES

Red Hat OpenStack Platform (RHOSP) runs services in containers on the undercloud and overcloud nodes. In certain situations, you might need to control the individual services on a host. This section contains information about some common commands you can run on a node to manage containerized services.

Listing containers and images

To list running containers, run the following command:

```
$ sudo podman ps
```

To include stopped or failed containers in the command output, add the **--all** option to the command:

```
$ sudo podman ps --all
```

To list container images, run the following command:

```
$ sudo podman images
```

Inspecting container properties

To view the properties of a container or container images, use the **podman inspect** command. For example, to inspect the **keystone** container, run the following command:

```
$ sudo podman inspect keystone
```

Managing containers with Systemd services

Previous versions of OpenStack Platform managed containers with Docker and its daemon. Now, the Systemd services interface manages the lifecycle of the containers. Each container is a service and you run Systemd commands to perform specific operations for each container.

> **NOTE**
>
> It is not recommended to use the Podman CLI to stop, start, and restart containers because Systemd applies a restart policy. Use Systemd service commands instead.

To check a container status, run the **systemctl status** command:

```
$ sudo systemctl status tripleo_keystone
● tripleo_keystone.service - keystone container
   Loaded: loaded (/etc/systemd/system/tripleo_keystone.service; enabled; vendor preset: disabled)
   Active: active (running) since Fri 2019-02-15 23:53:18 UTC; 2 days ago
 Main PID: 29012 (podman)
   CGroup: /system.slice/tripleo_keystone.service
           └─29012 /usr/bin/podman start -a keystone
```

To stop a container, run the **systemctl stop** command:

```
$ sudo systemctl stop tripleo_keystone
```

To start a container, run the **systemctl start** command:

```
$ sudo systemctl start tripleo_keystone
```

To restart a container, run the **systemctl restart** command:

```
$ sudo systemctl restart tripleo_keystone
```

Because no daemon monitors the containers status, Systemd automatically restarts most containers in these situations:

- Clean exit code or signal, such as running **podman stop** command.

- Unclean exit code, such as the podman container crashing after a start.

- Unclean signals.

- Timeout if the container takes more than 1m 30s to start.

For more information about Systemd services, see the **systemd.service** documentation.

> **NOTE**
>
> Any changes to the service configuration files within the container revert after restarting the container. This is because the container regenerates the service configuration based on files on the local file system of the node in **/var/lib/config-data/puppet-generated/**. For example, if you edit **/etc/keystone/keystone.conf** within the **keystone** container and restart the container, the container regenerates the configuration using **/var/lib/config-data/puppet-generated/keystone/etc/keystone/keystone.conf** on the local file system of the node, which overwrites any the changes that were made within the container before the restart.

Monitoring podman containers with Systemd timers

The Systemd timers interface manages container health checks. Each container has a timer that runs a service unit that executes health check scripts.

To list all OpenStack Platform containers timers, run the **systemctl list-timers** command and limit the output to lines containing **tripleo**:

```
$ sudo systemctl list-timers | grep tripleo
Mon 2019-02-18 20:18:30 UTC  1s left       Mon 2019-02-18 20:17:26 UTC  1min 2s ago
tripleo_nova_metadata_healthcheck.timer           tripleo_nova_metadata_healthcheck.service
Mon 2019-02-18 20:18:34 UTC  5s left       Mon 2019-02-18 20:17:23 UTC  1min 5s ago
tripleo_keystone_healthcheck.timer               tripleo_keystone_healthcheck.service
Mon 2019-02-18 20:18:35 UTC  6s left       Mon 2019-02-18 20:17:13 UTC  1min 15s ago
tripleo_memcached_healthcheck.timer              tripleo_memcached_healthcheck.service
(...)
```

To check the status of a specific container timer, run the **systemctl status** command for the healthcheck service:

```
$ sudo systemctl status tripleo_keystone_healthcheck.service
● tripleo_keystone_healthcheck.service - keystone healthcheck
   Loaded: loaded (/etc/systemd/system/tripleo_keystone_healthcheck.service; disabled; vendor preset: disabled)
   Active: inactive (dead) since Mon 2019-02-18 20:22:46 UTC; 22s ago
  Process: 115581 ExecStart=/usr/bin/podman exec keystone /openstack/healthcheck (code=exited, status=0/SUCCESS)
 Main PID: 115581 (code=exited, status=0/SUCCESS)

Feb 18 20:22:46 undercloud.localdomain systemd[1]: Starting keystone healthcheck...
Feb 18 20:22:46 undercloud.localdomain podman[115581]: {"versions": {"values": [{"status": "stable",
"updated": "2019-01-22T00:00:00Z", "..."}]}]}}
Feb 18 20:22:46 undercloud.localdomain podman[115581]: 300 192.168.24.1:35357 0.012 seconds
Feb 18 20:22:46 undercloud.localdomain systemd[1]: Started keystone healthcheck.
```

To stop, start, restart, and show the status of a container timer, run the relevant **systemctl** command against the **.timer** Systemd resource. For example, to check the status of the **tripleo_keystone_healthcheck.timer** resource, run the following command:

```
$ sudo systemctl status tripleo_keystone_healthcheck.timer
● tripleo_keystone_healthcheck.timer - keystone container healthcheck
   Loaded: loaded (/etc/systemd/system/tripleo_keystone_healthcheck.timer; enabled; vendor preset: disabled)
   Active: active (waiting) since Fri 2019-02-15 23:53:18 UTC; 2 days ago
```

If the healthcheck service is disabled but the timer for that service is present and enabled, it means that the check is currently timed out, but will be run according to timer. You can also start the check manually.
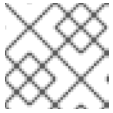
> **NOTE**
>
> The **podman ps** command does not show the container health status.

Checking container logs

Red Hat OpenStack Platform 17.1 logs all standard output (stdout) from all containers, and standard errors (stderr) consolidated inone single file for each container in **/var/log/containers/stdout**.

The host also applies log rotation to this directory, which prevents huge files and disk space issues.

In case a container is replaced, the new container outputs to the same log file, because **podman** uses the container name instead of container ID.

You can also check the logs for a containerized service with the **podman logs** command. For example, to view the logs for the **keystone** container, run the following command:

```
$ sudo podman logs keystone
```

Accessing containers

To enter the shell for a containerized service, use the **podman exec** command to launch **/bin/bash**. For example, to enter the shell for the **keystone** container, run the following command:

```
$ sudo podman exec -it keystone /bin/bash
```

To enter the shell for the **keystone** container as the root user, run the following command:

```
$ sudo podman exec --user 0 -it <NAME OR ID> /bin/bash
```

To exit the container, run the following command:

```
# exit
```

9.3. MODIFYING THE OVERCLOUD ENVIRONMENT

You can modify the overcloud to add additional features or alter existing operations.

Procedure

1. To modify the overcloud, make modifications to your custom environment files and heat templates, then rerun the **openstack overcloud deploy** command from your initial overcloud creation. For example, if you created an overcloud using Section 7.3, "Configuring and deploying the overcloud", rerun the following command:

```
$ source ~/stackrc
(undercloud) $ openstack overcloud deploy --templates \
 -e ~/templates/overcloud-baremetal-deployed.yaml \
```

```
    -e ~/templates/network-environment.yaml \
    -e ~/templates/storage-environment.yaml \
    --ntp-server pool.ntp.org
```

Director checks the **overcloud** stack in heat, and then updates each item in the stack with the environment files and heat templates. Director does not recreate the overcloud, but rather changes the existing overcloud.

> **IMPORTANT**
>
> Removing parameters from custom environment files does not revert the parameter value to the default configuration. You must identify the default value from the core heat template collection in **/usr/share/openstack-tripleo-heat-templates** and set the value in your custom environment file manually.

2. If you want to include a new environment file, add it to the **openstack overcloud deploy** command with the `-e` option. For example:

```
$ source ~/stackrc
(undercloud) $ openstack overcloud deploy --templates \
    -e ~/templates/new-environment.yaml \
    -e ~/templates/network-environment.yaml \
    -e ~/templates/storage-environment.yaml \
    -e ~/templates/overcloud-baremetal-deployed.yaml \
    --ntp-server pool.ntp.org
```

This command includes the new parameters and resources from the environment file into the stack.

> **IMPORTANT**
>
> It is not advisable to make manual modifications to the overcloud configuration because director might overwrite these modifications later.

9.4. IMPORTING VIRTUAL MACHINES INTO THE OVERCLOUD

You can migrate virtual machines from an existing OpenStack environment to your Red Hat OpenStack Platform (RHOSP) environment.

Procedure

1. On the existing OpenStack environment, create a new image by taking a snapshot of a running server and download the image:

```
$ openstack server image create --name <image_name> <instance_name>
$ openstack image save --file <exported_vm.qcow2> <image_name>
```

- Replace **<instance_name>** with the name of the instance.

- Replace **<image_name>** with the name of the new image.

- Replace **<exported_vm.qcow2>** with the name of the exported virtual machine.

2. Copy the exported image to the undercloud node:

```
$ scp exported_vm.qcow2 stack@192.168.0.2:~/.
```

3. Log in to the undercloud as the **stack** user.

4. Source the **overcloudrc** credentials file:

```
$ source ~/overcloudrc
```

5. Upload the exported image into the overcloud:

```
(overcloud) $ openstack image create --disk-format qcow2  -file <exported_vm.qcow2> --container-format bare <image_name>
```

6. Launch a new instance:

```
(overcloud) $ openstack server create  --key-name default --flavor m1.demo --image imported_image --nic net-id=net_id <instance_name>
```

> **IMPORTANT**
>
> You can use these commands to copy each virtual machine disk from the existing OpenStack environment to the new Red Hat OpenStack Platform. QCOW snapshots lose their original layering system.

9.5. LAUNCHING THE EPHEMERAL HEAT PROCESS

In previous versions of Red Hat OpenStack Platform (RHOSP) a system-installed Heat process was used to install the overcloud. Now, we use ephemeral Heat to install the overcloud meaning that the **heat-api** and **heat-engine** processes are started on demand by the **deployment**, **update**, and **upgrade** commands.

Previously, you used the **openstack stack** command to create and manage stacks. This command is no longer available by default. For troubleshooting and debugging purposes, for example if the stack should fail, you must first launch the ephemeral Heat process to use the **openstack stack** commands.

Use the **openstack overcloud tripleo launch heat** command to enable ephemeral heat outside of a deployment.

Procedure

1. Launch the ephemeral Heat process:

```
(undercloud)$ openstack tripleo launch heat --heat-dir /home/stack/overcloud-deploy/<overcloud>/heat-launcher --restore-db
```

- Replace **<overcloud>** with the name of your overcloud stack.
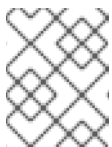
> **NOTE**
>
> The command exits after launching the Heat process, and the Heat process continues to run in the background as a Podman pod.

2. Verify that the **ephemeral-heat** process is running:

```
(undercloud)$ sudo podman pod ps
POD ID        NAME          STATUS    CREATED       INFRA ID      # OF CONTAINERS
958b141609b2  ephemeral-heat  Running     2 minutes ago  44447995dbcf  3
```

3. Export the **OS_CLOUD** environment:

```
(undercloud)$ export OS_CLOUD=heat
```

4. List the installed stacks:

```
(undercloud)$ openstack stack list
+--------------------------------------+------------+---------+----------------+---------------------+--------------+
| ID                                   | Stack Name | Project | Stack Status   | Creation Time       | Updated Time |
+--------------------------------------+------------+---------+----------------+---------------------+--------------+
| 761e2a54-c6f9-4e0f-abe6-c8e0ad51a76c | overcloud  | admin   | CREATE_COMPLETE | 2022-08-29T20:48:37Z | None         |
+--------------------------------------+------------+---------+----------------+---------------------+--------------+
```

You can debug with commands such as **openstack stack environment show** and **openstack stack resource list**.

5. After you have finished debugging, stop the ephemeral Heat process:

```
(undercloud)$ openstack tripleo launch heat --kill
```

> **NOTE**
>
> Sometimes, exporting the heat environment fails. This can happen when other credentials, such as **overcloudrc**, are in use. In this case unset the existing environment and source the heat environment.
>
> ```
> (overcloud)$ unset OS_CLOUD
> (overcloud)$ unset OS_PROJECT_NAME
> (overcloud)$ unset OS_PROJECT_DOMAIN_NAME
> (overcloud)$ unset OS_USER_DOMAIN_NAME
> (overcloud)$ OS_AUTH_TYPE=none
> (overcloud)$ OS_ENDPOINT=http://127.0.0.1:8006/v1/admin
> (overcloud)$ export OS_CLOUD=heat
> ```

9.6. RUNNING THE DYNAMIC INVENTORY SCRIPT

You can run Ansible-based automation in your Red Hat OpenStack Platform (RHOSP) environment. Use the **tripleo-ansible-inventory.yaml** inventory file located in the **/home/stack/overcloud-deploy/<stack>** directory to run ansible plays or ad-hoc commands.

NOTE

If you want to run an Ansible playbook or an Ansible ad-hoc command on the undercloud, you must use the **/home/stack/tripleo-deploy/undercloud/tripleo-ansible-inventory.yaml** inventory file.

Procedure

1. To view your inventory of nodes, run the following Ansible ad-hoc command:

   (undercloud) $ ansible -i ./overcloud-deploy/<stack>/tripleo-ansible-inventory.yaml all --list
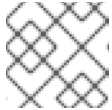
   NOTE

   Replace stack with the name of your deployed overcloud stack.

2. To execute Ansible playbooks on your environment, run the **ansible** command and include the full path to inventory file using the **-i** option. For example:

   (undercloud) $ ansible <hosts> -i ./overcloud-deploy/tripleo-ansible-inventory.yaml <playbook> <options>

   - Replace **<hosts>** with the type of hosts that you want to use to use:

     - **controller** for all Controller nodes

     - **compute** for all Compute nodes

     - **overcloud** for all overcloud child nodes. For example, **controller** and **compute** nodes

     - **"*"** for all nodes

   - Replace **<options>** with additional Ansible options.

     - Use the **--ssh-extra-args='-o StrictHostKeyChecking=no'** option to bypass confirmation on host key checking.

     - Use the **-u [USER]** option to change the SSH user that executes the Ansible automation. The default SSH user for the overcloud is automatically defined using the **ansible_ssh_user** parameter in the dynamic inventory. The **-u** option overrides this parameter.

     - Use the **-m [MODULE]** option to use a specific Ansible module. The default is **command**, which executes Linux commands.

     - Use the **-a [MODULE_ARGS]** option to define arguments for the chosen module.
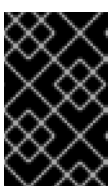
   IMPORTANT

   Custom Ansible automation on the overcloud is not part of the standard overcloud stack. Subsequent execution of the **openstack overcloud deploy** command might override Ansible-based configuration for OpenStack Platform services on overcloud nodes.

9.7. REMOVING AN OVERCLOUD STACK

You can delete an overcloud stack and unprovision all the stack nodes.

> **NOTE**
>
> Deleting your overcloud stack does not erase all the overcloud data. If you need to erase all the overcloud data, contact Red Hat support.

Procedure

1. Log in to the undercloud host as the **stack** user.

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

   ```
   $ source ~/stackrc
   ```

3. Retrieve a list of all the nodes in your stack and their current status:

   ```
   (undercloud)$ openstack baremetal node list
   +-------------------------------------+------------+--------------------------------------+-------------+--------------------+-------------+
   | UUID                                | Name       | Instance UUID                        | Power State | Provisioning State | Maintenance |
   +-------------------------------------+------------+--------------------------------------+-------------+--------------------+-------------+
   | 92ae71b0-3c31-4ebb-b467-6b5f6b0caac7 | compute-0  | 059fb1a1-53ea-4060-9a47-09813de28ea1 | power on    | active             | False       |
   | 9d6f955e-3d98-4d1a-9611-468761cebabf | compute-1  | e73a4b50-9579-4fe1-bd1a-556a2c8b504f | power on    | active             | False       |
   | 8a686fc1-1381-4238-9bf3-3fb16eaec6ab | controller-0 | 6d69e48d-10b4-45dd-9776-155a9b8ad575 | power on    | active             | False       |
   | eb8083cc-5f8f-405f-9b0c-14b772ce4534 | controller-1 | 1f836ac0-a70d-4025-88a3-bbe0583b4b8e | power on    | active             | False       |
   | a6750f1f-8901-41d6-b9f1-f5d6a10a76c7 | controller-2 | e2edd028-cea6-4a98-955e-5c392d91ed46 | power on    | active             | False       |
   +-------------------------------------+------------+--------------------------------------+-------------+--------------------+-------------+
   ```

4. Delete the overcloud stack and unprovision the nodes and networks:

   ```
   (undercloud)$ openstack overcloud delete -b <node_definition_file> \
   --networks-file <networks_definition_file> --network-ports <stack>
   ```

   - Replace **<node_definition_file>** with the name of your node definition file, for example, **overcloud-baremetal-deploy.yaml**.

   - Replace **<networks_definition_file>** with the name of your networks definition file, for example, **network_data_v2.yaml**.

   - Replace **<stack>** with the name of the stack that you want to delete. If not specified, the default stack is **overcloud**.

5. Confirm that you want to delete the overcloud:

   ```
   Are you sure you want to delete this overcloud [y/N]?
   ```

6. Wait for the overcloud to delete and the nodes and networks to unprovision.

7. Confirm that the bare-metal nodes have been unprovisioned:

```
(undercloud) [stack@undercloud-0 ~]$ openstack baremetal node list
+--------------------------------------+------------+---------------+-------------+--------------------+-------------+
| UUID                                 | Name       | Instance UUID | Power State | Provisioning State | Maintenance |
+--------------------------------------+------------+---------------+-------------+--------------------+-------------+
| 92ae71b0-3c31-4ebb-b467-6b5f6b0caac7 | compute-0  | None          | power off   | available          | False       |
| 9d6f955e-3d98-4d1a-9611-468761cebabf | compute-1  | None          | power off   | available          | False       |
| 8a686fc1-1381-4238-9bf3-3fb16eaec6ab | controller-0 | None        | power off   | available          | False       |
| eb8083cc-5f8f-405f-9b0c-14b772ce4534 | controller-1 | None        | power off   | available          | False       |
| a6750f1f-8901-41d6-b9f1-f5d6a10a76c7 | controller-2 | None        | power off   | available          | False       |
+--------------------------------------+------------+---------------+-------------+--------------------+-------------+
```

8. Remove the stack directories:

```
$ rm -rf ~/overcloud-deploy/<stack>
$ rm -rf ~/config-download/<stack>
```

NOTE

The directory paths for your stack might be different from the default if you used the **--output-dir** and **--working-dir** options when deploying the overcloud with the **openstack overcloud deploy** command.

9.8. MANAGING LOCAL DISK PARTITION SIZES

If your local disk partitions continue to fill up after you have optimized the configuration of your partition sizes, then perform one of the following tasks:

- Manually delete files from the affected partitions.

- Add a new physical disk and add it to the LVM volume group. For more information, see Configuring and managing logical volumes.

- Overprovision the partition to use the remaining spare disk space. This option is possible because the default whole disk overcloud image, **overcloud-hardened-uefi-full.qcow2**, is backed by a thin pool. For more information on thin-provisioned logical volumes, see Creating and managing thin provisioned volumes (thin volumes) in the RHEL *Configuring and managing local volumes* guide.

> **WARNING**
>
> Only use overprovisioning when it is not possible to manually delete files or add a new physical disk. Overprovisioning can fail during the write operation if there is insufficient free physical space.

> **NOTE**
>
> Adding a new disk and overprovisioning the partition require a support exception. Contact the Red Hat Customer Experience and Engagement team to discuss a support exception, if applicable, or other options.

CHAPTER 10. SCALING OVERCLOUD NODES

If you want to add or remove nodes after the creation of the overcloud, you must update the overcloud.

NOTE

Ensure that your bare metal nodes are not in maintenance mode before you begin scaling out or removing an overcloud node.

Use the following table to determine support for scaling each node type:

Table 10.1. Scale support for each node type

| Node type	Scale up?	Scale down?	Notes
Controller	N	N	You can replace Controller nodes using the procedures in Chapter 11, *Replacing Controller nodes*.
Compute	Y	Y	
Ceph Storage nodes	Y	N	You must have at least 1 Ceph Storage node from the initial overcloud creation.
Object Storage nodes	Y	Y	

IMPORTANT

Ensure that you have at least 10 GB free space before you scale the overcloud. This free space accommodates image conversion and caching during the node provisioning process.

10.1. ADDING NODES TO THE OVERCLOUD

You can add more nodes to your overcloud.
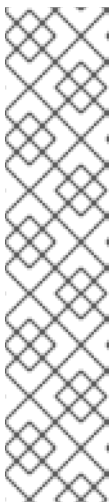
> **NOTE**
>
> A fresh installation of Red Hat OpenStack Platform (RHOSP) does not include certain updates, such as security errata and bug fixes. As a result, if you are scaling up a connected environment that uses the Red Hat Customer Portal or Red Hat Satellite Server, RPM updates are not applied to new nodes. To apply the latest updates to the overcloud nodes, you must do one of the following:
>
> - Complete an overcloud update of the nodes after the scale-out operation.
>
> - Use the **virt-customize** tool to modify the packages to the base overcloud image before the scale-out operation. For more information, see the Red Hat Knowledgebase solution Modifying the Red Hat Linux OpenStack Platform Overcloud Image with virt-customize.

Procedure

1. Create a new JSON file called **newnodes.json** that contains details of the new node that you want to register:

```
{
  "nodes":[
    {
      "mac":[
        "dd:dd:dd:dd:dd:dd"
      ],
      "cpu":"4",
      "memory":"6144",
      "disk":"40",
      "arch":"x86_64",
      "pm_type":"ipmi",
      "pm_user":"admin",
      "pm_password":"p@55w0rd!",
      "pm_addr":"192.02.24.207"
    },
    {
      "mac":[
        "ee:ee:ee:ee:ee:ee"
      ],
      "cpu":"4",
      "memory":"6144",
      "disk":"40",
      "arch":"x86_64",
      "pm_type":"ipmi",
      "pm_user":"admin",
      "pm_password":"p@55w0rd!",
      "pm_addr":"192.02.24.208"
    }
  ]
}
```

2. Log in to the undercloud host as the **stack** user.

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

```
$ source ~/stackrc
```

4. Register the new nodes:

```
$ openstack overcloud node import newnodes.json
```

5. Launch the introspection process for each new node:

```
$ openstack overcloud node introspect \
--provide <node_1> [<node_2>] [<node_n>]
```

- Use the **--provide** option to reset all the specified nodes to an **available** state after introspection.

- Replace **<node_1>**, **<node_2>**, and all nodes up to **<node_n>** with the UUID of each node that you want to introspect.

6. Configure the image properties for each new node:

```
$ openstack overcloud node configure <node>
```

10.2. SCALING UP BARE-METAL NODES

To increase the count of bare-metal nodes in an existing overcloud, increment the node count in the **overcloud-baremetal-deploy.yaml** file and redeploy the overcloud.

Prerequisites

- The new bare-metal nodes are registered, introspected, and available for provisioning and deployment. For more information, see Registering nodes for the overcloud and Creating an inventory of the bare-metal node hardware.

Procedure

1. Log in to the undercloud host as the **stack** user.

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

```
$ source ~/stackrc
```

3. Open the **overcloud-baremetal-deploy.yaml** node definition file that you use to provision your bare-metal nodes.

4. Increment the **count** parameter for the roles that you want to scale up. For example, the following configuration increases the Object Storage node count to 4:

```
- name: Controller
  count: 3
- name: Compute
  count: 10
- name: ObjectStorage
  count: 4
```

5. Optional: Configure predictive node placement for the new nodes. For example, use the following configuration to provision a new Object Storage node on **node03**:

```
- name: ObjectStorage
  count: 4
  instances:
  - hostname: overcloud-objectstorage-0
    name: node00
  - hostname: overcloud-objectstorage-1
    name: node01
  - hostname: overcloud-objectstorage-2
    name: node02
  - hostname: overcloud-objectstorage-3
    name: node03
```

6. Optional: Define any other attributes that you want to assign to your new nodes. For more information about the properties you can use to configure node attributes in your node definition file, see Bare-metal node provisioning attributes.

7. If you use the Object Storage service (swift) and the whole disk overcloud image, **overcloud-hardened-uefi-full**, configure the size of the **/srv** partition based on the size of your disk and your storage requirements for **/var** and **/srv**. For more information, see Configuring whole disk partitions for the Object Storage service.

8. Provision the overcloud nodes:

```
$ openstack overcloud node provision \
  --stack <stack> \
  --network-config \
  --output <deployment_file> \
  /home/stack/templates/overcloud-baremetal-deploy.yaml
```

- Replace **<stack>** with the name of the stack for which the bare-metal nodes are provisioned. If not specified, the default is **overcloud**.

- Include the **--network-config** argument to provide the network definitions to the **cli-overcloud-node-network-config.yaml** Ansible playbook.

- Replace **<deployment_file>** with the name of the heat environment file to generate for inclusion in the deployment command, for example **/home/stack/templates/overcloud-baremetal-deployed.yaml**.

> **NOTE**
>
> If you upgraded from Red Hat OpenStack Platform 16.2 to 17.1, you must include the YAML file that you created or updated during the upgrade process in the **openstack overcloud node provision** command. For example, use the **/home/stack/tripleo-[stack]-baremetal-deploy.yaml** file instead of the **/home/stack/templates/overcloud-baremetal-deployed.yaml** file. For more information, see Performing the overcloud adoption and preparation in *Framework for upgrades (16.2 to 17.1)*.

9. Monitor the provisioning progress in a separate terminal. When provisioning is successful, the node state changes from **available** to **active**:

```
$ watch openstack baremetal node list
```

10. Add the generated **overcloud-baremetal-deployed.yaml** file to the stack with your other environment files and deploy the overcloud:

```
$ openstack overcloud deploy --templates \
    -e [your environment files] \
    -e /home/stack/templates/overcloud-baremetal-deployed.yaml \
    --deployed-server \
    --disable-validations \
    ...
```

10.3. SCALING DOWN BARE-METAL NODES

To scale down the number of bare-metal nodes in your overcloud, tag the nodes that you want to delete from the stack in the node definition file, redeploy the overcloud, and then delete the bare-metal node from the overcloud.

Prerequisites

- A successful undercloud installation. For more information, see Installing director on the undercloud.

- A successful overcloud deployment. For more information, see Configuring a basic overcloud with pre-provisioned nodes.

- If you are replacing an Object Storage node, replicate data from the node you are removing to the new replacement node. Wait for a replication pass to finish on the new node. Check the replication pass progress in the **/var/log/swift/swift.log** file. When the pass finishes, the Object Storage service (swift) adds entries to the log similar to the following example:

  ```
  Mar 29 08:49:05 localhost object-server: Object replication complete.
  Mar 29 08:49:11 localhost container-server: Replication run OVER
  Mar 29 08:49:13 localhost account-server: Replication run OVER
  ```

Procedure

1. Log in to the undercloud host as the **stack** user.

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

   ```
   $ source ~/stackrc
   ```

3. Decrement the **count** parameter in the **overcloud-baremetal-deploy.yaml** file, for the roles that you want to scale down.

4. Define the **hostname** and **name** of each node that you want to remove from the stack, if they are not already defined in the **instances** attribute for the role.

5. Add the attribute **provisioned: false** to the node that you want to remove. For example, to remove the node **overcloud-objectstorage-1** from the stack, include the following snippet in your **overcloud-baremetal-deploy.yaml** file:

```
- name: ObjectStorage
  count: 3
  instances:
  - hostname: overcloud-objectstorage-0
    name: node00
  - hostname: overcloud-objectstorage-1
    name: node01
    # Removed from cluster due to disk failure
    provisioned: false
  - hostname: overcloud-objectstorage-2
    name: node02
  - hostname: overcloud-objectstorage-3
    name: node03
```

After you redeploy the overcloud, the nodes that you define with the **provisioned: false** attribute are no longer present in the stack. However, these nodes are still running in a provisioned state.

> **NOTE**
>
> To remove a node from the stack temporarily, deploy the overcloud with the attribute **provisioned: false** and then redeploy the overcloud with the attribute **provisioned: true** to return the node to the stack.

6. Delete the node from the overcloud:

   ```
   $ openstack overcloud node delete \
     --stack <stack> \
     --baremetal-deployment \
      /home/stack/templates/overcloud-baremetal-deploy.yaml
   ```

   - Replace **<stack>** with the name of the stack for which the bare-metal nodes are provisioned. If not specified, the default is **overcloud**.

     > **NOTE**
     >
     > Do not include the nodes that you want to remove from the stack as command arguments in the **openstack overcloud node delete** command.

7. Delete the ironic node:

   ```
   $ openstack baremetal node delete <IRONIC_NODE_UUID>
   ```

   Replace **IRONIC_NODE_UUID** with the UUID of the node.

8. Provision the overcloud nodes to generate an updated heat environment file for inclusion in the deployment command:

   ```
   $ openstack overcloud node provision \
     --stack <stack> \
     --output <deployment_file> \
     /home/stack/templates/overcloud-baremetal-deploy.yaml
   ```

- Replace **<deployment_file>** with the name of the heat environment file to generate for inclusion in the deployment command, for example **/home/stack/templates/overcloud-baremetal-deployed.yaml**.

9. Add the **overcloud-baremetal-deployed.yaml** file generated by the provisioning command to the stack with your other environment files, and deploy the overcloud:

```
$ openstack overcloud deploy \
...
-e /usr/share/openstack-tripleo-heat-templates/environments \
-e /home/stack/templates/overcloud-baremetal-deployed.yaml \
--deployed-server \
--disable-validations \
...
```

10.4. REPLACING RED HAT CEPH STORAGE NODES

You can use director to replace Red Hat Ceph Storage nodes in a director-created cluster. For more information, see the Deploying Red Hat Ceph Storage and Red Hat OpenStack Platform together with director guide.

10.5. USING SKIP DEPLOY IDENTIFIER

During a stack update operation puppet, by default, reapplies all manifests. This can result in a time consuming operation, which may not be required.

To override the default operation, use the **skip-deploy-identifier** option.

```
openstack overcloud deploy --skip-deploy-identifier
```

Use this option if you do not want the deployment command to generate a unique identifier for the **DeployIdentifier** parameter. The software configuration deployment steps only trigger if there is an actual change to the configuration. Use this option with caution and only if you are confident that you do not need to run the software configuration, such as scaling out certain roles.

> **NOTE**
>
> If there is a change to the puppet manifest or hierdata, puppet will reapply all manifests even when **--skip-deploy-identifier** is specified.

10.6. BLACKLISTING NODES

You can exclude overcloud nodes from receiving an updated deployment. This is useful in scenarios where you want to scale new nodes and exclude existing nodes from receiving an updated set of parameters and resources from the core heat template collection. This means that the blacklisted nodes are isolated from the effects of the stack operation.

Use the **DeploymentServerBlacklist** parameter in an environment file to create a blacklist.

Setting the blacklist

The **DeploymentServerBlacklist** parameter is a list of server names. Write a new environment file, or add the parameter value to an existing custom environment file and pass the file to the deployment command:

```
parameter_defaults:
  DeploymentServerBlacklist:
    - overcloud-compute-0
    - overcloud-compute-1
    - overcloud-compute-2
```

NOTE

The server names in the parameter value are the names according to OpenStack Orchestration (heat), not the actual server hostnames.

Include this environment file with your **openstack overcloud deploy** command:

```
$ source ~/stackrc
(undercloud) $ openstack overcloud deploy --templates \
  -e server-blacklist.yaml \
  [OTHER OPTIONS]
```

Heat blacklists any servers in the list from receiving updated heat deployments. After the stack operation completes, any blacklisted servers remain unchanged. You can also power off or stop the **os-collect-config** agents during the operation.

WARNING

- Exercise caution when you blacklist nodes. Only use a blacklist if you fully understand how to apply the requested change with a blacklist in effect. It is possible to create a hung stack or configure the overcloud incorrectly when you use the blacklist feature. For example, if cluster configuration changes apply to all members of a Pacemaker cluster, blacklisting a Pacemaker cluster member during this change can cause the cluster to fail.

- Do not use the blacklist during update or upgrade procedures. Those procedures have their own methods for isolating changes to particular servers.

- When you add servers to the blacklist, further changes to those nodes are not supported until you remove the server from the blacklist. This includes updates, upgrades, scale up, scale down, and node replacement. For example, when you blacklist existing Compute nodes while scaling out the overcloud with new Compute nodes, the blacklisted nodes miss the information added to **/etc/hosts** and **/etc/ssh/ssh_known_hosts**. This can cause live migration to fail, depending on the destination host. The Compute nodes are updated with the information added to **/etc/hosts** and **/etc/ssh/ssh_known_hosts** during the next overcloud deployment where they are no longer blacklisted. Do not modify the **/etc/hosts** and **/etc/ssh/ssh_known_hosts** files manually. To modify the **/etc/hosts** and **/etc/ssh/ssh_known_hosts** files, run the overcloud deploy command as described in the *Clearing the Blacklist* section.

Clearing the blacklist

To clear the blacklist for subsequent stack operations, edit the **DeploymentServerBlacklist** to use an empty array:

```
parameter_defaults:
  DeploymentServerBlacklist: []
```

> **WARNING**
>
> Do not omit the **DeploymentServerBlacklist** parameter. If you omit the parameter, the overcloud deployment uses the previously saved value.

CHAPTER 11. REPLACING CONTROLLER NODES

In certain circumstances a Controller node in a high availability cluster might fail. In these situations, you must remove the node from the cluster and replace it with a new Controller node.

Complete the steps in this section to replace a Controller node. The Controller node replacement process involves running the **openstack overcloud deploy** command to update the overcloud with a request to replace a Controller node.

> **IMPORTANT**
>
> The following procedure applies only to high availability environments. Do not use this procedure if you are using only one Controller node.

11.1. PREPARING FOR CONTROLLER REPLACEMENT

Before you replace an overcloud Controller node, it is important to check the current state of your Red Hat OpenStack Platform environment. Checking the current state can help avoid complications during the Controller replacement process. Use the following list of preliminary checks to determine if it is safe to perform a Controller node replacement. Run all commands for these checks on the undercloud.

Procedure

1. Check the current status of the **overcloud** stack on the undercloud:

   ```
   $ source stackrc
   $ openstack overcloud status
   ```

   Only continue if the **overcloud** stack has a deployment status of **DEPLOY_SUCCESS**.

2. Install the database client tools:

   ```
   $ sudo dnf -y install mariadb
   ```

3. Configure root user access to the database:

   ```
   $ sudo cp /var/lib/config-data/puppet-generated/mysql/root/.my.cnf /root/.
   ```

4. Perform a backup of the undercloud databases:

   ```
   $ mkdir /home/stack/backup
   $ sudo mysqldump --all-databases --quick --single-transaction | gzip > /home/stack/backup/dump_db_undercloud.sql.gz
   ```

5. Check that your undercloud contains 10 GB free storage to accommodate for image caching and conversion when you provision the new node:

   ```
   $ df -h
   ```

6. If you are reusing the IP address for the new controller node, ensure that you delete the port used by the old controller:

   ```
   $ openstack port delete <port>
   ```

7. Check the status of Pacemaker on the running Controller nodes. For example, if 192.168.0.47 is the IP address of a running Controller node, use the following command to view the Pacemaker status:

```
$ ssh tripleo-admin@192.168.0.47 'sudo pcs status'
```

The output shows all services that are running on the existing nodes and those that are stopped on the failed node.

8. Check the following parameters on each node of the overcloud MariaDB cluster:

   - **wsrep_local_state_comment: Synced**

   - **wsrep_cluster_size: 2**
     Use the following command to check these parameters on each running Controller node. In this example, the Controller node IP addresses are 192.168.0.47 and 192.168.0.46:

   ```
   $ for i in 192.168.0.46 192.168.0.47 ; do echo "*** $i ***" ; ssh tripleo-admin@$i "sudo
   podman exec \$(sudo podman ps --filter name=galera-bundle -q) mysql -e \"SHOW
   STATUS LIKE 'wsrep_local_state_comment'; SHOW STATUS LIKE
   'wsrep_cluster_size';\""; done
   ```

9. Check the RabbitMQ status. For example, if 192.168.0.47 is the IP address of a running Controller node, use the following command to view the RabbitMQ status:

   ```
   $ ssh tripleo-admin@192.168.0.47 "sudo podman exec \$(sudo podman ps -f
   name=rabbitmq-bundle -q) rabbitmqctl cluster_status"
   ```

   The **running_nodes** key should show only the two available nodes and not the failed node.

10. If fencing is enabled, disable it. For example, if 192.168.0.47 is the IP address of a running Controller node, use the following command to check the status of fencing:

   ```
   $ ssh tripleo-admin@192.168.0.47 "sudo pcs property show stonith-enabled"
   ```

   Run the following command to disable fencing:

   ```
   $ ssh tripleo-admin@192.168.0.47 "sudo pcs property set stonith-enabled=false"
   ```

11. Login to the failed Controller node and stop all the **nova_*** containers that are running:

   ```
   $ sudo systemctl stop tripleo_nova_api.service
   $ sudo systemctl stop tripleo_nova_api_cron.service
   $ sudo systemctl stop tripleo_nova_conductor.service
   $ sudo systemctl stop tripleo_nova_metadata.service
   $ sudo systemctl stop tripleo_nova_scheduler.service
   ```

12. Optional: If you are using the Bare Metal Service (ironic) as the virt driver, you must manually update the service entries in your cell database for any bare metal instances whose **instances.host** is set to the controller that you are removing. Contact Red Hat Support for assistance.

> **NOTE**
>
> This manual update of the cell database when using Bare Metal Service (ironic) as the virt driver is a temporary workaround to ensure the nodes are rebalanced, until BZ2017980 is complete.

11.2. REMOVING A CEPH MONITOR DAEMON

If your Controller node is running a Ceph monitor service, complete the following steps to remove the **ceph-mon** daemon.

> **NOTE**
>
> Adding a new Controller node to the cluster also adds a new Ceph monitor daemon automatically.

Procedure

1. Connect to the Controller node that you want to replace:

   ```
   $ ssh tripleo-admin@192.168.0.47
   ```

2. List the Ceph mon services:

   ```
   $ sudo systemctl --type=service | grep ceph
   ceph-4cf401f9-dd4c-5cda-9f0a-fa47fbf12b31@crash.controller-0.service         loaded active
   running Ceph crash.controller-0 for 4cf401f9-dd4c-5cda-9f0a-fa47fbf12b31
     ceph-4cf401f9-dd4c-5cda-9f0a-fa47fbf12b31@mgr.controller-0.mufglq.service     loaded
   active running Ceph mgr.controller-0.mufglq for 4cf401f9-dd4c-5cda-9f0a-fa47fbf12b31
     ceph-4cf401f9-dd4c-5cda-9f0a-fa47fbf12b31@mon.controller-0.service         loaded active
   running Ceph mon.controller-0 for 4cf401f9-dd4c-5cda-9f0a-fa47fbf12b31
     ceph-4cf401f9-dd4c-5cda-9f0a-fa47fbf12b31@rgw.rgw.controller-0.ikaevh.service loaded
   active running Ceph rgw.rgw.controller-0.ikaevh for 4cf401f9-dd4c-5cda-9f0a-fa47fbf12b31
   ```

3. Stop the Ceph mon service:

   ```
   $ sudo systemtctl stop ceph-4cf401f9-dd4c-5cda-9f0a-fa47fbf12b31@mon.controller-0.service
   ```

4. Disable the Ceph mon service:

   ```
   $ sudo systemctl disable ceph-4cf401f9-dd4c-5cda-9f0a-fa47fbf12b31@mon.controller-0.service
   ```

5. Disconnect from the Controller node that you want to replace.

6. Use SSH to connect to another Controller node in the same cluster:

   ```
   $ ssh tripleo-admin@192.168.0.46
   ```

7. The Ceph specification file is modified and applied later in this procedure, to manipulate the file you must export it:

```
$ sudo cephadm shell --ceph orch ls --export > spec.yaml
```

8. Remove the monitor from the cluster:

```
$ sudo cephadm shell -- ceph mon remove controller-0
  removing mon.controller-0 at [v2:172.23.3.153:3300/0,v1:172.23.3.153:6789/0], there will be
2 monitors
```

9. Disconnect from the Controller node and log back into the Controller node you are removing from the cluster:

```
$ ssh tripleo-admin@192.168.0.47
```

10. List the Ceph mgr services:

```
$ sudo systemctl --type=service | grep ceph
ceph-4cf401f9-dd4c-5cda-9f0a-fa47fbf12b31@crash.controller-0.service        loaded active
running Ceph crash.controller-0 for 4cf401f9-dd4c-5cda-9f0a-fa47fbf12b31
  ceph-4cf401f9-dd4c-5cda-9f0a-fa47fbf12b31@mgr.controller-0.mufglq.service     loaded
active running Ceph mgr.controller-0.mufglq for 4cf401f9-dd4c-5cda-9f0a-fa47fbf12b31
  ceph-4cf401f9-dd4c-5cda-9f0a-fa47fbf12b31@rgw.rgw.controller-0.ikaevh.service loaded
active running Ceph rgw.rgw.controller-0.ikaevh for 4cf401f9-dd4c-5cda-9f0a-fa47fbf12b31
```

11. Stop the Ceph mgr service:

```
$ sudo systemctl stop ceph-4cf401f9-dd4c-5cda-9f0a-fa47fbf12b31@mgr.controller-0.mufglq.service
```

12. Disable the Ceph mgr service:

```
$ sudo systemctl disable ceph-4cf401f9-dd4c-5cda-9f0a-fa47fbf12b31@mgr.controller-0.mufglq.service
```

13. Start a **cephadm** shell:

```
$ sudo cephadm shell
```

14. Verify that the Ceph mgr service for the Controller node is removed from the cluster:

```
$ ceph -s
cluster:
    id:     b9b53581-d590-41ac-8463-2f50aa985001
    health: HEALTH_OK

  services:
    mon: 2 daemons, quorum controller-2,controller-1 (age 2h)
    mgr: controller-2(active, since 20h), standbys: controller1-1
    osd: 15 osds: 15 up (since 3h), 15 in (since 3h)

  data:
    pools:   3 pools, 384 pgs
```

```
    objects: 32 objects, 88 MiB
    usage:   16 GiB used, 734 GiB / 750 GiB avail
    pgs:     384 active+clean
```

The node is not listed if the Ceph mgr service is successfully removed.

15. Export the Red Hat Ceph Storage specification:

    ```
    $ ceph orch ls --export > spec.yaml
    ```

16. In the **spec.yaml** specification file, remove all instances of the host, for example **controller-0**, from the **service_type: mon** and **service_type: mgr**.

17. Reapply the Red Hat Ceph Storage specification:

    ```
    $ ceph orch apply -i spec.yaml
    ```

18. Verify that no Ceph daemons remain on the removed host:

    ```
    $ ceph orch ps controller-0
    ```

    > **NOTE**
    >
    > If daemons are present, use the following command to remove them:
    >
    > ```
    > $ ceph orch host drain controller-0
    > ```
    >
    > Prior to running the **ceph orch host drain** command, backup the contents of **/etc/ceph**. Restore the contents after running the **ceph orch host drain** command. You must back up prior to running the **ceph orch host drain** command until https://bugzilla.redhat.com/show_bug.cgi?id=2153827 is resolved.

19. Remove the **controller-0** host from the Red Hat Ceph Storage cluster:

    ```
    $ ceph orch host rm controller-0
      Removed host 'controller-0'
    ```

20. Exit the cephadm shell:

    ```
    $ exit
    ```

Additional Resources

- For more information on controlling Red Hat Ceph Storage services with systemd, see Understanding process management for Ceph .

- For more information on editing and applying Red Hat Ceph Storage specification files, see Deploying the Ceph monitor daemons using the service specification .

11.3. PREPARING THE CLUSTER FOR CONTROLLER NODE REPLACEMENT

Before you replace the node, ensure that Pacemaker is not running on the node and then remove that node from the Pacemaker cluster.

Procedure

1. To view the list of IP addresses for the Controller nodes, run the following command:

   ```
   (undercloud)$ metalsmith -c Hostname -c "IP Addresses" list
   +------------------------+-----------------------+
   | Hostname               | IP Addresses          |
   +------------------------+-----------------------+
   | overcloud-compute-0    | ctlplane=192.168.0.44 |
   | overcloud-controller-0 | ctlplane=192.168.0.47 |
   | overcloud-controller-1 | ctlplane=192.168.0.45 |
   | overcloud-controller-2 | ctlplane=192.168.0.46 |
   +------------------------+-----------------------+
   ```

2. Log in to the node and confirm the pacemaker status. If pacemaker is running, use the **pcs cluster** command to stop pacemaker. This example stops pacemaker on **overcloud-controller-0**:

   ```
   (undercloud) $ ssh tripleo-admin@192.168.0.45 "sudo pcs status | grep -w Online | grep -w overcloud-controller-0"
   (undercloud) $ ssh tripleo-admin@192.168.0.45 "sudo pcs cluster stop overcloud-controller-0"
   ```
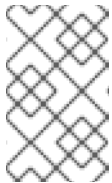
   > **NOTE**
   >
   > In the case that the node is physically unavailable or stopped, it is not necessary to perform the previous operation, as pacemaker is already stopped on that node.

3. After you stop Pacemaker on the node, delete the node from the pacemaker cluster. The following example logs in to **overcloud-controller-1** to remove **overcloud-controller-0**:

   ```
   (undercloud) $ ssh tripleo-admin@192.168.0.45 "sudo pcs cluster node remove overcloud-controller-0"
   ```

   If the node that that you want to replace is unreachable (for example, due to a hardware failure), run the **pcs** command with additional **--skip-offline** and **--force** options to forcibly remove the node from the cluster:

   ```
   (undercloud) $ ssh tripleo-admin@192.168.0.45 "sudo pcs cluster node remove overcloud-controller-0 --skip-offline --force"
   ```

4. After you remove the node from the pacemaker cluster, remove the node from the list of known hosts in pacemaker:

   ```
   (undercloud) $ ssh tripleo-admin@192.168.0.45 "sudo pcs host deauth overcloud-controller-0"
   ```

You can run this command whether the node is reachable or not.

5. To ensure that the new Controller node uses the correct STONITH fencing device after replacement, delete the devices from the node by entering the following command:

```
(undercloud) $ ssh tripleo-admin@192.168.0.45 "sudo pcs stonith delete <stonith_resource_name>"
```

- Replace **<stonith_resource_name>** with the name of the STONITH resource that corresponds to the node. The resource name uses the format **<resource_agent>-<host_mac>**. You can find the resource agent and the host MAC address in the **FencingConfig** section of the **fencing.yaml** file.

6. The overcloud database must continue to run during the replacement procedure. To ensure that Pacemaker does not stop Galera during this procedure, select a running Controller node and run the following command on the undercloud with the IP address of the Controller node:

```
(undercloud) $ ssh tripleo-admin@192.168.0.45 "sudo pcs resource unmanage galera-bundle"
```

7. Remove the OVN northbound database server for the replaced Controller node from the cluster:

a. Obtain the server ID of the OVN northbound database server to be replaced:

```
$ ssh tripleo-admin@<controller_ip> sudo podman exec ovn_cluster_north_db_server ovs-appctl -t /var/run/ovn/ovnnb_db.ctl cluster/status OVN_Northbound 2>/dev/null|grep -A4 Servers:
```

Replace **<controller_ip>** with the IP address of any active Controller node.

You should see output similar to the following:

```
Servers:
96da (96da at tcp:172.17.1.55:6643) (self) next_index=26063 match_index=26063 466b (466b at tcp:172.17.1.51:6643) next_index=26064 match_index=26063 last msg 2936 ms ago
ba77 (ba77 at tcp:172.17.1.52:6643) next_index=26064 match_index=26063 last msg 2936 ms ago
```

In this example, **172.17.1.55** is the internal IP address of the Controller node that is being replaced, so the northbound database server ID is **96da**.

b. Using the server ID you obtained in the preceding step, remove the OVN northbound database server by running the following command:

```
$ ssh tripleo-admin@172.17.1.52 sudo podman exec ovn_cluster_north_db_server ovs-appctl -t /var/run/ovn/ovnnb_db.ctl cluster/kick OVN_Northbound 96da
```

In this example, you would replace **172.17.1.52** with the IP address of any active Controller node, and replace **96da** with the server ID of the OVN northbound database server.

8. Remove the OVN southbound database server for the replaced Controller node from the cluster:

a. Obtain the server ID of the OVN southbound database server to be replaced:

```
$ ssh tripleo-admin@<controller_ip> sudo podman exec ovn_cluster_south_db_server
ovs-appctl -t /var/run/ovn/ovnsb_db.ctl cluster/status OVN_Southbound 2>/dev/null|grep
-A4 Servers:
```

Replace **<controller_ip>** with the IP address of any active Controller node.

You should see output similar to the following:

```
Servers:
e544 (e544 at tcp:172.17.1.55:6644) last msg 42802690 ms ago
17ca (17ca at tcp:172.17.1.51:6644) last msg 5281 ms ago
6e52 (6e52 at tcp:172.17.1.52:6644) (self)
```

In this example, **172.17.1.55** is the internal IP address of the Controller node that is being replaced, so the southbound database server ID is **e544**.

b. Using the server ID you obtained in the preceding step, remove the OVN southbound database server by running the following command:

```
$ ssh tripleo-admin@172.17.1.52 sudo podman exec ovn_cluster_south_db_server ovs-
appctl -t /var/run/ovn/ovnsb_db.ctl cluster/kick OVN_Southbound e544
```

In this example, you would replace **172.17.1.52** with the IP address of any active Controller node, and replace **e544** with the server ID of the OVN southbound database server.

9. Run the following clean up commands to prevent cluster rejoins.
Substitute **<replaced_controller_ip>** with the IP address of the Controller node that you are replacing:

```
$ ssh tripleo-admin@<replaced_controller_ip> sudo systemctl disable --now
tripleo_ovn_cluster_south_db_server.service tripleo_ovn_cluster_north_db_server.service
```

```
$ ssh tripleo-admin@<replaced_controller_ip> sudo rm -rfv /var/lib/openvswitch/ovn/.ovn*
/var/lib/openvswitch/ovn/ovn*.db
```

11.4. REPLACING A BOOTSTRAP CONTROLLER NODE

If you want to replace the Controller node that you use for bootstrap operations and keep the node name, complete the following steps to set the name of the bootstrap Controller node after the replacement process.

Procedure

1. Find the name of the bootstrap Controller node by running the following command:

```
ssh tripleo-admin@<controller_ip> "sudo hiera -c /etc/puppet/hiera.yaml
pacemaker_short_bootstrap_node_name"
```

- Replace **<controller_ip>** with the IP address of any active Controller node.

2. Check if your environment files include the **ExtraConfig** and **AllNodesExtraMapData** parameters. If the parameters are not set, create the following environment file ~/**templates/bootstrap-controller.yaml** and add the following content:

```
parameter_defaults:
  ExtraConfig:
    pacemaker_short_bootstrap_node_name: NODE_NAME
    mysql_short_bootstrap_node_name: NODE_NAME
  AllNodesExtraMapData:
    ovn_dbs_bootstrap_node_ip: NODE_IP
    ovn_dbs_short_bootstrap_node_name: NODE_NAME
```

- Replace **NODE_NAME** with the name of an existing Controller node that you want to use in bootstrap operations after the replacement process.

- Replace **NODE_IP** with the IP address mapped to the Controller named in **NODE_NAME**. To get the name, run the following command:

```
sudo hiera -c /etc/puppet/hiera.yaml ovn_dbs_node_ips
```

If your environment files already include the **ExtraConfig** and **AllNodesExtraMapData** parameters, add only the lines shown in this step.

For information about troubleshooting the bootstrap Controller node replacement, see the article .

11.5. UNPROVISION AND REMOVE CONTROLLER NODES

You can unprovision and remove Controller nodes.

Procedure

1. Source the **stackrc** file:

```
$ source ~/stackrc
```

2. Identify the UUID of the **overcloud-controller-0** node:

```
(undercloud)$ NODE=$(metalsmith -c UUID -f value show overcloud-controller-0)
```

3. Set the node to maintenance mode:

```
$ openstack baremetal node maintenance set $NODE
```

4. Copy the **overcloud-baremetal-deploy.yaml** file:

```
$ cp /home/stack/templates/overcloud-baremetal-deploy.yaml
/home/stack/templates/unprovision_controller-0.yaml
```

5. In the **unprovision_controller-0.yaml** file, lower the Controller count to unprovision the Controller node that you are replacing. In this example, the count is reduced from **3** to **2**. Move the **controller-0** node to the **instances** dictionary and set the **provisioned** parameter to **false**:

```
- name: Controller
  count: 2
  hostname_format: controller-%index%
  defaults:
    resource_class: BAREMETAL.controller
    networks:
      [ ... ]
  instances:
  - hostname: controller-0
    name: <IRONIC_NODE_UUID_or_NAME>
    provisioned: false
- name: Compute
  count: 2
  hostname_format: compute-%index%
  defaults:
    resource_class: BAREMETAL.compute
    networks:
      [ ... ]
```

6. Run the **node unprovision** command:

```
$ openstack overcloud node delete \
  --stack overcloud \
  --baremetal-deployment /home/stack/templates/unprovision_controller-0.yaml
```

```
The following nodes will be unprovisioned:
+-------------+-----------------------+--------------------------------------+
| hostname    | name                  | id                                   |
+-------------+-----------------------+--------------------------------------+
| controller-0 | baremetal-35400-leaf1-2 | b0d5abf7-df28-4ae7-b5da-9491e84c21ac |
+-------------+-----------------------+--------------------------------------+

Are you sure you want to unprovision these overcloud nodes and ports [y/N]?
```

7. Optional: Delete the ironic node:

```
$ openstack baremetal node delete <IRONIC_NODE_UUID>
```

- Replace **IRONIC_NODE_UUID** with the UUID of the node.

11.6. DEPLOYING A NEW CONTROLLER NODE TO THE OVERCLOUD

To deploy a new controller node to the overcloud complete the following steps.

Prerequisites

- The new Controller node must be registered, inspected, and tagged ready for provisioning. For more information, see Provisioning bare metal overcloud nodes

Procedure

1. Log in to the undercloud host as the **stack** user.

2. Source the **stackrc** undercloud credentials file:
   $ source ~/stackrc

3. Provision the overcloud with the original **overcloud-baremetal-deploy.yaml** environment file:

```
$ openstack overcloud node provision
  --stack overcloud
  --network-config
  --output /home/stack/templates/overcloud-baremetal-deployed.yaml
  /home/stack/templates/overcloud-baremetal-deploy.yaml
```
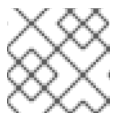
NOTE

If you want to use the same scheduling, placement, or IP addresses you can edit the **overcloud-baremetal-deploy.yaml** environment file. Set the hostname, name, and networks for the new controller-0 instance in the **instances** section. For example:

```
- name: Controller
  count: 3
  hostname_format: controller-%index%
  defaults:
    resource_class: BAREMETAL.controller
    networks:
    - network: external
      subnet: external_subnet
    - network: internal_api
      subnet: internal_api_subnet01
    - network: storage
      subnet: storage_subnet01
    - network: storage_mgmt
      subnet: storage_mgmt_subnet01
    - network: tenant
      subnet: tenant_subnet01
    network_config:
      template: templates/multiple_nics/multiple_nics_dvr.j2
      default_route_network:
      - external
  instances:
  - hostname: controller-0
    name: baremetal-35400-leaf1-2
    networks:
    - network: external
      subnet: external_subnet
      fixed_ip: 10.0.0.224
    - network: internal_api
      subnet: internal_api_subnet01
      fixed_ip: 172.17.0.97
    - network: storage
      subnet: storage_subnet01
      fixed_ip: 172.18.0.24
    - network: storage_mgmt
      subnet: storage_mgmt_subnet01
      fixed_ip: 172.19.0.129
    - network: tenant
      subnet: tenant_subnet01
      fixed_ip: 172.16.0.11
- name: Compute
  count: 2
  hostname_format: compute-%index%
  defaults:
    [ ... ]
```

When the node is provisioned, remove the **instances** section from the **overcloud-baremetal-deploy.yaml** file.

4. To create the **cephadm** user on the new Controller node, export a basic Ceph specification containing the new host information:

```
$ openstack overcloud ceph spec --stack overcloud \
  /home/stack/templates/overcloud-baremetal-deployed.yaml \
  -o ceph_spec_host.yaml
```

> **NOTE**
>
> If your environment uses a custom role, include the **--roles-data** option.

5. Add the **cephadm** user to the new Controller node:

```
$ openstack overcloud ceph user enable \
  --stack overcloud ceph_spec_host.yaml
```

6. Log in to the Controller node and add the new role to the Ceph cluster:

```
$ sudo cephadm shell \
  -- ceph orch host add controller-3 <IP_ADDRESS> <LABELS>
192.168.24.31 _admin mon mgr
Inferring fsid 4cf401f9-dd4c-5cda-9f0a-fa47fbf12b31
Using recent ceph image undercloud-0.ctlplane.redhat.local:8787/rh-
osbs/rhceph@sha256:3075e8708792ebd527ca14849b6af4a11256a3f881ab09b837d7af0f8b2
102ea
Added host 'controller-3' with addr '192.168.24.31'
```

- Replace <IP_ADDRESS> with the IP address of the Controller node.

- Replace <LABELS> with any required Ceph labels.

7. Re-run the **openstack overcloud deploy** command:

```
$ openstack overcloud deploy --stack overcloud --templates \
    -n /home/stack/templates/network_data.yaml \
    -r /home/stack/templates/roles_data.yaml \
    -e /home/stack/templates/overcloud-baremetal-deployed.yaml \
    -e /home/stack/templates/overcloud-networks-deployed.yaml \
    -e /home/stack/templates/overcloud-vips-deployed.yaml \
    -e /home/stack/templates/bootstrap_node.yaml \
    -e [ ... ]
```

> **NOTE**
>
> If the replacement Controller node is the bootstrap node, include the **bootstrap_node.yaml** environment file.

11.7. DEPLOYING CEPH SERVICES ON THE NEW CONTROLLER NODE

After you provision a new Controller node and the Ceph monitor services are running you can deploy the **mgr**, **rgw** and **osd** Ceph services on the Controller node.

Prerequisites

.

- The new Controller node is provisioned and is running Ceph monitor services.

Procedure

1. Modify the **spec.yml** environment file, replace the previous Controller node name with the new Controller node name:

   ```
   $ cephadm shell -- ceph orch ls --export > spec.yml
   ```

   > **NOTE**
   >
   > Do not use the basic Ceph environment file **ceph_spec_host.yaml** as it does not contain all necessary cluster information.

2. Apply the modified Ceph specification file:

   ```
   $ cat spec.yml | sudo cephadm shell -- ceph orch apply -i -
   Inferring fsid 4cf401f9-dd4c-5cda-9f0a-fa47fbf12b31
   Using recent ceph image undercloud-0.ctlplane.redhat.local:8787/rh-
   osbs/rhceph@sha256:3075e8708792ebd527ca14849b6af4a11256a3f881ab09b837d7af0f8b2
   102ea
   Scheduled crash update...
   Scheduled mgr update...
   Scheduled mon update...
   Scheduled osd.default_drive_group update...
   Scheduled rgw.rgw update...
   ```

3. Verify the visibility of the new monitor:

   ```
   $ sudo cephadm --ceph status
   ```

.

11.8. CLEANING UP AFTER CONTROLLER NODE REPLACEMENT

After you complete the node replacement, you can finalize the Controller cluster.

Procedure

1. Log into a Controller node.

2. Enable Pacemaker management of the Galera cluster and start Galera on the new node:

   ```
   [tripleo-admin@overcloud-controller-0 ~]$ sudo pcs resource refresh galera-bundle
   [tripleo-admin@overcloud-controller-0 ~]$ sudo pcs resource manage galera-bundle
   ```
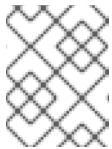
3. Enable fencing:

   ```
   [tripleo-admin@overcloud-controller-0 ~]$ sudo pcs property set stonith-enabled=true
   ```

4. Perform a final status check to ensure that the services are running correctly:

```
[tripleo-admin@overcloud-controller-0 ~]$ sudo pcs status
```

> **NOTE**
>
> If any services have failed, use the **pcs resource refresh** command to resolve
> and restart the failed services.

5. Exit to director:

```
[tripleo-admin@overcloud-controller-0 ~]$ exit
```

6. Source the **overcloudrc** file so that you can interact with the overcloud:

```
$ source ~/overcloudrc
```

7. Check the network agents in your overcloud environment:

```
(overcloud) $ openstack network agent list
```

8. If any agents appear for the old node, remove them:

```
(overcloud) $ for AGENT in $(openstack network agent list --host overcloud-controller-
1.localdomain -c ID -f value) ; do openstack network agent delete $AGENT ; done
```

9. If necessary, add your router to the L3 agent host on the new node. Use the following example
   command to add a router named **r1** to the L3 agent using the UUID 2d1c1dc1-d9d4-4fa9-b2c8-
   f29cd1a649d4:

```
(overcloud) $ openstack network agent add router --l3 2d1c1dc1-d9d4-4fa9-b2c8-
f29cd1a649d4 r1
```

10. Clean the cinder services.

   a. List the cinder services:

```
(overcloud) $ openstack volume service list
```

   b. Log in to a controller node, connect to the **cinder-api** container and use the **cinder-
      manage service remove** command to remove leftover services:

```
[tripleo-admin@overcloud-controller-0 ~]$ sudo podman exec -it cinder_api cinder-
manage service remove cinder-backup <host>
[tripleo-admin@overcloud-controller-0 ~]$ sudo podman exec -it cinder_api cinder-
manage service remove cinder-scheduler <host>
```

11. Clean the RabbitMQ cluster.

   a. Log into a Controller node.

   b. Use the **podman exec** command to launch bash, and verify the status of the RabbitMQ
      cluster:

```
[tripleo-admin@overcloud-controller-0 ~]$ sudo podman exec -it rabbitmq-bundle-
podman-0 bash
[root@overcloud-controller-0 /]$ rabbitmqctl cluster_status
```

c. Use the **rabbitmqctl** command to forget the replaced controller node:

```
[root@controller-0 /]$ rabbitmqctl forget_cluster_node <node_name>
```

12. If you replaced a bootstrap Controller node, you must remove the environment file
~/**templates**/**bootstrap-controller.yaml** after the replacement process, or delete the
pacemaker_short_bootstrap_node_name and **mysql_short_bootstrap_node_name**
parameters from your existing environment file. This step prevents director from attempting to
override the Controller node name in subsequent replacements. For more information, see
Replacing a bootstrap Controller node .

13. If you are using the Object Storage service (swift) on the overcloud, you must synchronize the
swift rings after updating the overcloud nodes. Use a script, similar to the following example, to
distribute ring files from a previously existing Controller node (Controller node 0 in this
example) to all Controller nodes and restart the Object Storage service containers on those
nodes:

```
#!/bin/sh
set -xe

SRC="tripleo-admin@overcloud-controller-0.ctlplane"
ALL="tripleo-admin@overcloud-controller-0.ctlplane tripleo-admin@overcloud-controller-
1.ctlplane tripleo-admin@overcloud-controller-2.ctlplane"
```

- Fetch the current set of ring files:

```
ssh "${SRC}" 'sudo tar -czvf - /var/lib/config-data/puppet-
generated/swift_ringbuilder/etc/swift/{*.builder,*.ring.gz,backups/*.builder}' > swift-
rings.tar.gz
```

- Upload rings to all nodes, put them into the correct place, and restart swift services:

```
for DST in ${ALL}; do
  cat swift-rings.tar.gz | ssh "${DST}" 'sudo tar -C / -xvzf -'
  ssh "${DST}" 'sudo podman restart swift_copy_rings'
  ssh "${DST}" 'sudo systemctl restart tripleo_swift*'
done
```

CHAPTER 12. REBOOTING NODES

You might need to reboot the nodes in the undercloud and overcloud.

> **NOTE**
>
> If you enabled instance HA (high availability) in your overcloud and if you need to shut down or reboot Compute nodes, see Chapter 3. Performing maintenance on the undercloud and overcloud with Instance HA in *Configuring high availability for instances* .

Use the following procedures to understand how to reboot different node types.

- If you reboot all nodes in one role, it is advisable to reboot each node individually. If you reboot all nodes in a role simultaneously, service downtime can occur during the reboot operation.

- If you reboot all nodes in your OpenStack Platform environment, reboot the nodes in the following sequential order:

Recommended node reboot order

1. Reboot the undercloud node.

2. Reboot Controller and other composable nodes.

3. Reboot standalone Ceph MON nodes.

4. Reboot Ceph Storage nodes.

5. Reboot Object Storage service (swift) nodes.

6. Reboot Compute nodes.

12.1. REBOOTING THE UNDERCLOUD NODE

Complete the following steps to reboot the undercloud node.

Procedure

1. Log in to the undercloud as the **stack** user.

2. Reboot the undercloud:

   ```
   $ sudo reboot
   ```

3. Wait until the node boots.

12.2. REBOOTING CONTROLLER AND COMPOSABLE NODES

Reboot Controller nodes and standalone nodes based on composable roles, and exclude Compute nodes and Ceph Storage nodes.

Procedure

1. Log in to the node that you want to reboot.

2. Optional: If the node uses Pacemaker resources, stop the cluster:

   ```
   [tripleo-admin@overcloud-controller-0 ~]$ sudo pcs cluster stop
   ```

3. Reboot the node:

   ```
   [tripleo-admin@overcloud-controller-0 ~]$ sudo reboot
   ```

4. Wait until the node boots.

Verification

1. Verify that the services are enabled.

   a. If the node uses Pacemaker services, check that the node has rejoined the cluster:

   ```
   [tripleo-admin@overcloud-controller-0 ~]$ sudo pcs status
   ```

   b. If the node uses Systemd services, check that all services are enabled:

   ```
   [tripleo-admin@overcloud-controller-0 ~]$ sudo systemctl status
   ```

   c. If the node uses containerized services, check that all containers on the node are active:

   ```
   [tripleo-admin@overcloud-controller-0 ~]$ sudo podman ps
   ```

12.3. REBOOTING STANDALONE CEPH MON NODES

Complete the following steps to reboot standalone Ceph MON nodes.

Procedure

1. Log in to a Ceph MON node.

2. Reboot the node:

   ```
   $ sudo reboot
   ```

3. Wait until the node boots and rejoins the MON cluster.

Repeat these steps for each MON node in the cluster.

12.4. REBOOTING A CEPH STORAGE (OSD) CLUSTER

Complete the following steps to reboot a cluster of Ceph Storage (OSD) nodes.

Prerequisites

- On a Ceph Monitor or Controller node that is running the **ceph-mon** service, check that the Red Hat Ceph Storage cluster status is healthy and the pg status is **active+clean**:

```
$ sudo cephadm -- shell ceph status
```

If the Ceph cluster is healthy, it returns a status of **HEALTH_OK**.

If the Ceph cluster status is unhealthy, it returns a status of **HEALTH_WARN** or **HEALTH_ERR**. For troubleshooting guidance, see the Red Hat Ceph Storage 5 Troubleshooting Guide or the Red Hat Ceph Storage 6 Troubleshooting Guide .

Procedure

1. Log in to a Ceph Monitor or Controller node that is running the **ceph-mon** service, and disable Ceph Storage cluster rebalancing temporarily:

   ```
   $ sudo cephadm shell -- ceph osd set noout
   $ sudo cephadm shell -- ceph osd set norebalance
   ```

   > **NOTE**
   >
   > If you have a multistack or distributed compute node (DCN) architecture, you must specify the Ceph cluster name when you set the **noout** and **norebalance** flags. For example: **sudo cephadm shell -c /etc/ceph/<cluster>.conf -k /etc/ceph/<cluster>.client.keyring**.

2. Select the first Ceph Storage node that you want to reboot and log in to the node.

3. Reboot the node:

   ```
   $ sudo reboot
   ```

4. Wait until the node boots.

5. Log in to the node and check the Ceph cluster status:

   ```
   $ sudo cephadm -- shell ceph status
   ```

   Check that the **pgmap** reports all **pgs** as normal (**active+clean**).

6. Log out of the node, reboot the next node, and check its status. Repeat this process until you have rebooted all Ceph Storage nodes.

7. When complete, log in to a Ceph Monitor or Controller node that is running the **ceph-mon** service and enable Ceph cluster rebalancing:

   ```
   $ sudo cephadm shell -- ceph osd unset noout
   $ sudo cephadm shell -- ceph osd unset norebalance
   ```

   > **NOTE**
   >
   > If you have a multistack or distributed compute node (DCN) architecture, you must specify the Ceph cluster name when you unset the **noout** and **norebalance** flags. For example: **sudo cephadm shell -c /etc/ceph/<cluster>.conf -k /etc/ceph/<cluster>.client.keyring**

8. Perform a final status check to verify that the cluster reports **HEALTH_OK**:

```
$ sudo cephadm shell ceph status
```

12.5. REBOOTING OBJECT STORAGE SERVICE (SWIFT) NODES

The following procedure reboots Object Storage service (swift) nodes. Complete the following steps for every Object Storage node in your cluster.

Procedure

1. Log in to an Object Storage node.

2. Reboot the node:

```
$ sudo reboot
```

3. Wait until the node boots.

4. Repeat the reboot for each Object Storage node in the cluster.

12.6. REBOOTING COMPUTE NODES

To ensure minimal downtime of instances in your Red Hat OpenStack Platform environment, the Migrating instances workflow outlines the steps you must complete to migrate instances from the Compute node that you want to reboot.

Migrating instances workflow

1. Decide whether to migrate instances to another Compute node before rebooting the node.

2. Select and disable the Compute node that you want to reboot so that it does not provision new instances.

3. Migrate the instances to another Compute node.

4. Reboot the empty Compute node.

5. Enable the empty Compute node.

Prerequisites

- Before you reboot the Compute node, you must decide whether to migrate instances to another Compute node while the node is rebooting.
  Review the list of migration constraints that you might encounter when you migrate virtual machine instances between Compute nodes. For more information, see Migration constraints in *Configuring the Compute service for instance creation* .

NOTE

If you have a Multi-RHEL environment, and you want to migrate virtual machines from a Compute node that is running RHEL 9.2 to a Compute node that is running RHEL 8.4, only cold migration is supported. For more information about cold migration, see Cold migrating an instance in *Configuring the Compute service for instance creation*.

- If you cannot migrate the instances, you can set the following core template parameters to control the state of the instances after the Compute node reboots:

   NovaResumeGuestsStateOnHostBoot

   Determines whether to return instances to the same state on the Compute node after reboot. When set to **False**, the instances remain down and you must start them manually. The default value is **False**.

   NovaResumeGuestsShutdownTimeout

   Number of seconds to wait for an instance to shut down before rebooting. It is not recommended to set this value to **0**. The default value is **300**.
   For more information about overcloud parameters and their usage, see Overcloud parameters.

Procedure

1. Log in to the undercloud as the **stack** user.

2. Retrieve a list of your Compute nodes to identify the host name of the node that you want to reboot:

   ```
   (undercloud)$ source ~/overcloudrc
   (overcloud)$ openstack compute service list
   ```

   Identify the host name of the Compute node that you want to reboot.

3. Disable the Compute service on the Compute node that you want to reboot:

   ```
   (overcloud)$ openstack compute service list
   (overcloud)$ openstack compute service set <hostname> nova-compute --disable
   ```

   - Replace **<hostname>** with the host name of your Compute node.

4. List all instances on the Compute node:

   ```
   (overcloud)$ openstack server list --host <hostname> --all-projects
   ```

5. Optional: To migrate the instances to another Compute node, complete the following steps:

   a. If you decide to migrate the instances to another Compute node, use one of the following commands:

      - To migrate the instance to a different host, run the following command:

         ```
         (overcloud) $ openstack server migrate <instance_id> --live <target_host> --wait
         ```

- Replace **<instance_id>** with your instance ID.

- Replace **<target_host>** with the host that you are migrating the instance to.

- Let **nova-scheduler** automatically select the target host:

  (overcloud) $ nova live-migration <instance_id>

- Live migrate all instances at once:

  $ nova host-evacuate-live <hostname>

> **NOTE**
>
> The **nova** command might cause some deprecation warnings, which are safe to ignore.

b. Wait until migration completes.

c. Confirm that the migration was successful:

  (overcloud) $ openstack server list --host <hostname> --all-projects

d. Continue to migrate instances until none remain on the Compute node.

6. Log in to the Compute node and reboot the node:

  [tripleo-admin@overcloud-compute-0 ~]$ sudo reboot

7. Wait until the node boots.

8. Re-enable the Compute node:

  $ source ~/overcloudrc
  (overcloud) $ openstack compute service set <hostname> nova-compute --enable

9. Check that the Compute node is enabled:

  (overcloud) $ openstack compute service list

CHAPTER 13. SHUTTING DOWN AND STARTING UP THE UNDERCLOUD AND OVERCLOUD

If you must perform maintenance on the undercloud and overcloud, you must shut down and start up the undercloud and overcloud nodes in a specific order to ensure minimal issues when your start your overcloud.

> **NOTE**
>
> If you enabled instance HA (high availability) in your overcloud and if you need to shut down or reboot Compute nodes, see Chapter 3. Performing maintenance on the undercloud and overcloud with Instance HA in the *Configuring high availability for instances*.

Prerequisites

- A running undercloud and overcloud

13.1. UNDERCLOUD AND OVERCLOUD SHUTDOWN ORDER

To shut down the Red Hat OpenStack Platform environment, you must shut down the overcloud and undercloud in the following order:

1. Shut down instances on overcloud Compute nodes

2. Shut down Compute nodes

3. Stop all high availability and OpenStack Platform services on Controller nodes

4. Shut down Ceph Storage nodes

5. Shut down Controller nodes

6. Shut down the undercloud

13.2. SHUTTING DOWN INSTANCES ON OVERCLOUD COMPUTE NODES

As a part of shutting down the Red Hat OpenStack Platform environment, shut down all instances on Compute nodes before shutting down the Compute nodes.

Prerequisites

- An overcloud with active Compute services

Procedure

1. Log in to the undercloud as the **stack** user.

2. Source the credentials file for your overcloud:

   ```
   $ source ~/overcloudrc
   ```

3. View running instances in the overcloud:

```
$ openstack server list --all-projects
```

4. Stop each instance in the overcloud:

```
$ openstack server stop <INSTANCE>
```

Repeat this step for each instance until you stop all instances in the overcloud.

13.3. SHUTTING DOWN COMPUTE NODES

As a part of shutting down the Red Hat OpenStack Platform environment, log in to and shut down each Compute node.

Prerequisites

- Shut down all instances on the Compute nodes

Procedure

1. Log in as the **root** user to a Compute node.

2. Shut down the node:

```
# shutdown -h now
```

3. Perform these steps for each Compute node until you shut down all Compute nodes.

13.4. STOPPING SERVICES ON CONTROLLER NODES

As a part of shutting down the Red Hat OpenStack Platform environment, stop services on the Controller nodes before shutting down the nodes. This includes Pacemaker and systemd services.

Prerequisites

- An overcloud with active Pacemaker services

Procedure

1. Log in as the **root** user to a Controller node.

2. Stop the Pacemaker cluster.

```
# pcs cluster stop --all
```

This command stops the cluster on all nodes.

3. Wait until the Pacemaker services stop and check that the services stopped.

   a. Check the Pacemaker status:

   ```
   # pcs status
   ```

b. Check that no Pacemaker services are running in Podman:

```
# podman ps --filter "name=.*-bundle.*"
```

4. Stop the Red Hat OpenStack Platform services:

```
# systemctl stop 'tripleo_*'
```

5. Wait until the services stop and check that services are no longer running in Podman:

```
# podman ps
```

13.5. SHUTTING DOWN CEPH STORAGE NODES

As a part of shutting down the Red Hat OpenStack Platform environment, disable Ceph Storage services then log in to and shut down each Ceph Storage node.

Prerequisites

- A healthy Ceph Storage cluster

- Ceph MON services are running on standalone Ceph MON nodes or on Controller nodes

Procedure

1. Log in as the **root** user to a node that runs Ceph MON services, such as a Controller node or a standalone Ceph MON node.

2. Check the health of the cluster. In the following example, the **podman** command runs a status check within a Ceph MON container on a Controller node:

```
# sudo podman exec -it ceph-mon-controller-0 ceph status
```

Ensure that the status is **HEALTH_OK**.

3. Set the **noout**, **norecover**, **norebalance**, **nobackfill**, **nodown**, and **pause** flags for the cluster. In the following example, the **podman** commands set these flags through a Ceph MON container on a Controller node:

```
# sudo podman exec -it ceph-mon-controller-0 ceph osd set noout
# sudo podman exec -it ceph-mon-controller-0 ceph osd set norecover
# sudo podman exec -it ceph-mon-controller-0 ceph osd set norebalance
# sudo podman exec -it ceph-mon-controller-0 ceph osd set nobackfill
# sudo podman exec -it ceph-mon-controller-0 ceph osd set nodown
# sudo podman exec -it ceph-mon-controller-0 ceph osd set pause
```

4. Shut down each Ceph Storage node:

a. Log in as the **root** user to a Ceph Storage node.

b. Shut down the node:

```
# shutdown -h now
```

c. Perform these steps for each Ceph Storage node until you shut down all Ceph Storage nodes.

5. Shut down any standalone Ceph MON nodes:

    a. Log in as the **root** user to a standalone Ceph MON node.

    b. Shut down the node:

        # shutdown -h now

    c. Perform these steps for each standalone Ceph MON node until you shut down all standalone Ceph MON nodes.

Additional resources

- "What is the procedure to shutdown and bring up the entire ceph cluster?"

13.6. SHUTTING DOWN CONTROLLER NODES

As a part of shutting down the Red Hat OpenStack Platform environment, log in to and shut down each Controller node.

Prerequisites

- Stop the Pacemaker cluster

- Stop all Red Hat OpenStack Platform services on the Controller nodes

Procedure

1. Log in as the **root** user to a Controller node.

2. Shut down the node:

        # shutdown -h now

3. Perform these steps for each Controller node until you shut down all Controller nodes.

13.7. SHUTTING DOWN THE UNDERCLOUD

As a part of shutting down the Red Hat OpenStack Platform environment, log in to the undercloud node and shut down the undercloud.

Prerequisites

- A running undercloud

Procedure

1. Log in to the undercloud as the **stack** user.

2. Shut down the undercloud:

```
$ sudo shutdown -h now
```

13.8. PERFORMING SYSTEM MAINTENANCE

After you completely shut down the undercloud and overcloud, perform any maintenance to the systems in your environment and then start up the undercloud and overcloud.

13.9. UNDERCLOUD AND OVERCLOUD STARTUP ORDER

To start the Red Hat OpenStack Platform environment, you must start the undercloud and overcloud in the following order:

1. Start the undercloud.

2. Start Controller nodes.

3. Start Ceph Storage nodes.

4. Start Compute nodes.

5. Start instances on overcloud Compute nodes.

13.10. STARTING THE UNDERCLOUD

As a part of starting the Red Hat OpenStack Platform environment, power on the undercloud node, log in to the undercloud, and check the undercloud services.

Prerequisites

- The undercloud is powered down.

Procedure

- Power on the undercloud and wait until the undercloud boots.

Verification

1. Log in to the undercloud host as the **stack** user.

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

   ```
   $ source ~/stackrc
   ```

3. Check the services on the undercloud:

   ```
   $ systemctl list-units 'tripleo_*'
   ```

4. Validate the static inventory file named **tripleo-ansible-inventory.yaml**:

   ```
   $ validation run --group pre-introspection -i <inventory_file>
   ```

- Replace **<inventory_file>** with the name and location of the Ansible inventory file, for example, ~/**tripleo-deploy/undercloud/tripleo-ansible-inventory.yaml**.

> **NOTE**
>
> When you run a validation, the **Reasons** column in the output is limited to 79 characters. To view the validation result in full, view the validation log files.

5. Check that all services and containers are active and healthy:

   ```
   $ validation run --validation service-status --limit undercloud -i <inventory_file>
   ```

Additional resources

- [Using the validation framework](#)

13.11. STARTING CONTROLLER NODES

As a part of starting the Red Hat OpenStack Platform environment, power on each Controller node and check the non-Pacemaker services on the node.

Prerequisites

- The Controller nodes are powered down.

Procedure

- Power on each Controller node.

Verification

1. Log in to each Controller node as the **root** user.

2. Check the services on the Controller node:

   ```
   $ systemctl -t service
   ```

   Only non-Pacemaker based services are running.

3. Wait until the Pacemaker services start and check that the services started:

   ```
   $ pcs status
   ```

   > **NOTE**
   >
   > If your environment uses Instance HA, the Pacemaker resources do not start until you start the Compute nodes or perform a manual unfence operation with the **pcs stonith confirm <compute_node>** command. You must run this command on each Compute node that uses Instance HA.

13.12. STARTING CEPH STORAGE NODES

As a part of starting the Red Hat OpenStack Platform environment, power on the Ceph MON and Ceph Storage nodes and enable Ceph Storage services.

Prerequisites

- A powered down Ceph Storage cluster

- Ceph MON services are enabled on powered down standalone Ceph MON nodes or on powered on Controller nodes

Procedure

1. If your environment has standalone Ceph MON nodes, power on each Ceph MON node.

2. Power on each Ceph Storage node.

3. Log in as the **root** user to a node that runs Ceph MON services, such as a Controller node or a standalone Ceph MON node.

4. Check the status of the cluster nodes. In the following example, the **podman** command runs a status check within a Ceph MON container on a Controller node:

   ```
   # sudo podman exec -it ceph-mon-controller-0 ceph status
   ```

   Ensure that each node is powered on and connected.

5. Unset the **noout**, **norecover**, **norebalance**, **nobackfill**, **nodown** and **pause** flags for the cluster. In the following example, the **podman** commands unset these flags through a Ceph MON container on a Controller node:

   ```
   # sudo podman exec -it ceph-mon-controller-0 ceph osd unset noout
   # sudo podman exec -it ceph-mon-controller-0 ceph osd unset norecover
   # sudo podman exec -it ceph-mon-controller-0 ceph osd unset norebalance
   # sudo podman exec -it ceph-mon-controller-0 ceph osd unset nobackfill
   # sudo podman exec -it ceph-mon-controller-0 ceph osd unset nodown
   # sudo podman exec -it ceph-mon-controller-0 ceph osd unset pause
   ```

Verification

1. Check the health of the cluster. In the following example, the **podman** command runs a status check within a Ceph MON container on a Controller node:

   ```
   # sudo podman exec -it ceph-mon-controller-0 ceph status
   ```

   Ensure the status is **HEALTH_OK**.

Additional resources

- "What is the procedure to shutdown and bring up the entire ceph cluster?"

13.13. STARTING COMPUTE NODES

As a part of starting the Red Hat OpenStack Platform environment, power on each Compute node and check the services on the node.

Prerequisites

- Powered down Compute nodes

Procedure

1. Power on each Compute node.

Verification

1. Log in to each Compute as the **root** user.

2. Check the services on the Compute node:

   ```
   $ systemctl -t service
   ```

13.14. STARTING INSTANCES ON OVERCLOUD COMPUTE NODES

As a part of starting the Red Hat OpenStack Platform environment, start the instances on on Compute nodes.

Prerequisites

- An active overcloud with active nodes

Procedure

1. Log in to the undercloud as the **stack** user.

2. Source the credentials file for your overcloud:

   ```
   $ source ~/overcloudrc
   ```

3. View running instances in the overcloud:

   ```
   $ openstack server list --all-projects
   ```

4. Start an instance in the overcloud:

   ```
   $ openstack server start <INSTANCE>
   ```

CHAPTER 14. TROUBLESHOOTING DIRECTOR ERRORS

Errors can occur at certain stages of the director processes. This section contains some information about diagnosing common problems.

14.1. TROUBLESHOOTING NODE REGISTRATION

Issues with node registration usually occur due to issues with incorrect node details. In these situations, validate the template file containing your node details and correct the imported node details.

Procedure

1. Source the **stackrc** file:

   ```
   $ source ~/stackrc
   ```

2. Run the node import command with the **--validate-only** option. This option validates your node template without performing an import:

   ```
   (undercloud) $ openstack overcloud node import --validate-only ~/nodes.json
   Waiting for messages on queue 'tripleo' with no timeout.

   Successfully validated environment file
   ```

3. To fix incorrect details with imported nodes, run the **openstack baremetal** commands to update node details. The following example shows how to change networking details:

   a. Identify the assigned port UUID for the imported node:

      ```
      $ source ~/stackrc
      (undercloud) $ openstack baremetal port list --node [NODE UUID]
      ```

   b. Update the MAC address:

      ```
      (undercloud) $ openstack baremetal port set --address=[NEW MAC] [PORT UUID]
      ```

   c. Configure a new IPMI address on the node:

      ```
      (undercloud) $ openstack baremetal node set --driver-info ipmi_address=[NEW IPMI ADDRESS] [NODE UUID]
      ```

14.2. TROUBLESHOOTING HARDWARE INTROSPECTION

The Bare Metal Provisioning inspector service, **ironic-inspector**, times out after a default one-hour period if the inspection RAM disk does not respond. The timeout might indicate a bug in the inspection RAM disk, but usually the timeout occurs due to an environment misconfiguration.

You can diagnose and resolve common environment misconfiguration issues to ensure the introspection process runs to completion.

Procedure

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

   ```
   $ source ~/stackrc
   ```

2. Ensure that your nodes are in a **manageable** state. The introspection does not inspect nodes in an **available** state, which is meant for deployment. If you want to inspect nodes that are in an **available** state, change the node status to **manageable** state before introspection:

   ```
   (undercloud)$ openstack baremetal node manage <node_uuid>
   ```

3. To configure temporary access to the introspection RAM disk during introspection debugging, use the **sshkey** parameter to append your public SSH key to the **kernel** configuration in the **/httpboot/inspector.ipxe** file:

   ```
   kernel http://192.2.0.1:8088/agent.kernel ipa-inspection-callback-
   url=http://192.168.0.1:5050/v1/continue ipa-inspection-collectors=default,extra-hardware,logs
   systemd.journald.forward_to_console=yes BOOTIF=${mac} ipa-debug=1 ipa-inspection-
   benchmarks=cpu,mem,disk selinux=0 sshkey="<public_ssh_key>"
   ```

4. Run the introspection on the node:

   ```
   (undercloud)$ openstack overcloud node introspect <node_uuid> --provide
   ```

   Use the **--provide** option to change the node state to **available** after the introspection completes.

5. Identify the IP address of the node from the **dnsmasq** logs:

   ```
   (undercloud)$ sudo tail -f /var/log/containers/ironic-inspector/dnsmasq.log
   ```

6. If an error occurs, access the node using the root user and temporary access details:

   ```
   $ ssh root@192.168.24.105
   ```

   Access the node during introspection to run diagnostic commands and troubleshoot the introspection failure.

7. To stop the introspection process, run the following command:

   ```
   (undercloud)$ openstack baremetal introspection abort <node_uuid>
   ```

   You can also wait until the process times out.
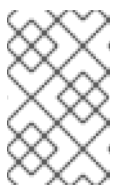
   > **NOTE**
   >
   > Red Hat OpenStack Platform director retries introspection three times after the initial abort. Run the **openstack baremetal introspection abort** command at each attempt to abort the introspection completely.

14.3. TROUBLESHOOTING OVERCLOUD CREATION AND DEPLOYMENT

The initial creation of the overcloud occurs with the OpenStack Orchestration (heat) service. If an overcloud deployment fails, use the OpenStack clients and service log files to diagnose the failed deployment.

Procedure

1. Source the **stackrc** file:

   ```
   $ source ~/stackrc
   ```

2. Launch the ephemeral Heat process:

   ```
   (undercloud)$ openstack tripleo launch heat --heat-dir /home/stack/overcloud-deploy/overcloud/heat-launcher --restore-db
   (undercloud)$ export OS_CLOUD=heat
   ```

3. View the details of the failure:

   ```
   (undercloud)$ openstack stack failures list <overcloud> --long
   ```

   - Replace **<overcloud>** with the name of your overcloud.

4. Identify the stacks that failed:

   ```
   (undercloud)$ openstack stack list --nested --property status=FAILED
   ```

5. Remove the ephemeral Heat process from the undercloud:

   ```
   (undercloud)$ openstack tripleo launch heat --kill
   ```

14.4. TROUBLESHOOTING NODE PROVISIONING

The OpenStack Orchestration (heat) service controls the provisioning process. If node provisioning fails, use the OpenStack clients and service log files to diagnose the issues.

Procedure

1. Source the **stackrc** file:

   ```
   $ source ~/stackrc
   ```

2. Check the bare metal service to see all registered nodes and their current status:

   ```
   (undercloud) $ openstack baremetal node list

   +----------+------+---------------+-------------+-----------------+-------------+
   | UUID     | Name | Instance UUID | Power State | Provision State | Maintenance |
   +----------+------+---------------+-------------+-----------------+-------------+
   | f1e261...| None | None          | power off   | available       | False       |
   | f0b8c1...| None | None          | power off   | available       | False       |
   +----------+------+---------------+-------------+-----------------+-------------+
   ```

All nodes available for provisioning should have the following states set:

- Maintenance set to **False**.

- Provision State set to **available** before provisioning.

3. If a node does not have **Maintenance** set to **False** or **Provision State** set to **available**, then use the following table to identify the problem and the solution:

Problem	Cause	Solution
Maintenance sets itself to **True** automatically.	Director cannot access the power management for the nodes.	Check the credentials for node power management.
Provision State is set to **available** but nodes do not provision.	The problem occurred before bare-metal deployment started.	Check that the node hardware details are within the requirements.
Provision State is set to **wait call-back** for a node.	The node provisioning process has not yet finished for this node.	Wait until this status changes. Otherwise, connect to the virtual console of the node and check the output.
Provision State is **active** and Power State is **power on** but the nodes do not respond.	The node provisioning has finished successfully and there is a problem during the post-deployment configuration step.	Diagnose the node configuration process. Connect to the virtual console of the node and check the output.
Provision State is **error** or **deploy failed**.	Node provisioning has failed.	View the bare metal node details with the **openstack baremetal node show** command and check the **last_error** field, which contains error description.

Additional resources

- Bare-metal node provisioning states

14.5. TROUBLESHOOTING IP ADDRESS CONFLICTS DURING PROVISIONING

Introspection and deployment tasks fail if the destination hosts are allocated an IP address that is already in use. To prevent these failures, you can perform a port scan of the Provisioning network to determine whether the discovery IP range and host IP range are free.

Procedure

1. Install **nmap**:

   ```
   $ sudo dnf install nmap
   ```

2. Use **nmap** to scan the IP address range for active addresses. This example scans the 192.168.24.0/24 range, replace this with the IP subnet of the Provisioning network (using CIDR bitmask notation):

   ```
   $ sudo nmap -sn 192.168.24.0/24
   ```

3. Review the output of the **nmap** scan. For example, you should see the IP address of the undercloud, and any other hosts that are present on the subnet:

   ```
   $ sudo nmap -sn 192.168.24.0/24

   Starting Nmap 6.40 ( http://nmap.org ) at 2015-10-02 15:14 EDT
   Nmap scan report for 192.168.24.1
   Host is up (0.00057s latency).
   Nmap scan report for 192.168.24.2
   Host is up (0.00048s latency).
   Nmap scan report for 192.168.24.3
   Host is up (0.00045s latency).
   Nmap scan report for 192.168.24.5
   Host is up (0.00040s latency).
   Nmap scan report for 192.168.24.9
   Host is up (0.00019s latency).
   Nmap done: 256 IP addresses (5 hosts up) scanned in 2.45 seconds
   ```

   If any of the active IP addresses conflict with the IP ranges in undercloud.conf, you must either change the IP address ranges or release the IP addresses before you introspect or deploy the overcloud nodes.

14.6. TROUBLESHOOTING OVERCLOUD CONFIGURATION

Red Hat OpenStack Platform director uses Ansible to configure the overcloud. Complete the following steps to diagnose Ansible playbook errors (**config-download**) on the overcloud.

Procedure

1. Ensure that the **stack** user has access to the files in the ~/**config-download/overcloud** directory on the **undercloud**:

   ```
   $ sudo setfacl -R -m u:stack:rwx ~/config-download/overcloud
   ```

2. Change to the working directory for the **config-download** files:

   ```
   $ cd ~/config-download/overcloud
   ```

3. Search the **ansible.log** file for the point of failure:

   ```
   $ less ansible.log
   ```

   Make a note of the step that failed.

4. Find the step that failed in the **config-download** playbooks within the working directory to identify the action that occurred.

14.7. TROUBLESHOOTING CONTAINER CONFIGURATION

Red Hat OpenStack Platform director uses **podman** to manage containers and **puppet** to create container configuration. This procedure shows how to diagnose a container when errors occur.

Accessing the host

1. Source the **stackrc** file:

   ```
   $ source ~/stackrc
   ```

2. Get the IP address of the node with the container failure.

   ```
   (undercloud) $ metalsmith list
   ```

3. Log in to the node:

   ```
   (undercloud) $ ssh tripleo-admin@192.168.24.60
   ```

Identifying failed containers

1. View all containers:

   ```
   $ sudo podman ps --all
   ```

   Identify the failed container. The failed container usually exits with a non-zero status.

Checking container logs

1. Each container retains standard output from its main process. Use this output as a log to help determine what actually occurs during a container run. For example, to view the log for the **keystone** container, run the following command:

   ```
   $ sudo podman logs keystone
   ```

   In most cases, this log contains information about the cause of a container failure.

2. The host also retains the **stdout** log for the failed service. You can find the **stdout** logs in **/var/log/containers/stdouts/**. For example, to view the log for a failed **keystone** container, run the following command:

   ```
   $ cat /var/log/containers/stdouts/keystone.log
   ```

Inspecting containers

In some situations, you might need to verify information about a container. For example, use the following command to view **keystone** container data:

```
$ sudo podman inspect keystone
```

This command returns a JSON object containing low-level configuration data. You can pipe the output to the **jq** command to parse specific data. For example, to view the container mounts for the **keystone** container, run the following command:

```
$ sudo podman inspect keystone | jq .[0].Mounts
```

You can also use the **--format** option to parse data to a single line, which is useful for running commands against sets of container data. For example, to recreate the options used to run the **keystone** container, use the following **inspect** command with the **--format** option:

```
$ sudo podman inspect --format='{{range .Config.Env}} -e "{{.}}" {{end}} {{range .Mounts}} -v {{.Source}}:{{.Destination}}:{{ join .Options "," }}{{end}} -ti {{.Config.Image}}' keystone
```

> **NOTE**
>
> The **--format** option uses Go syntax to create queries.

Use these options in conjunction with the **podman run** command to recreate the container for troubleshooting purposes:

```
$ OPTIONS=$( sudo podman inspect --format='{{range .Config.Env}} -e "{{.}}" {{end}} {{range .Mounts}} -v {{.Source}}:{{.Destination}}{{if .Mode}}:{{.Mode}}{{end}}{{end}} -ti {{.Config.Image}}' keystone )
$ sudo podman run --rm $OPTIONS /bin/bash
```

Running commands in a container

In some cases, you might need to obtain information from within a container through a specific Bash command. In this situation, use the following **podman** command to execute commands within a running container. For example, run the **podman exec** command to run a command inside the **keystone** container:

```
$ sudo podman exec -ti keystone <COMMAND>
```
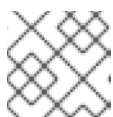
> **NOTE**
>
> The **-ti** options run the command through an interactive pseudoterminal.

- Replace **<COMMAND>** with the command you want to run. For example, each container has a health check script to verify the service connection. You can run the health check script for **keystone** with the following command:

```
$ sudo podman exec -ti keystone /openstack/healthcheck
```

To access the container shell, run **podman exec** using **/bin/bash** as the command you want to run inside the container:

```
$ sudo podman exec -ti keystone /bin/bash
```

Viewing a container filesystem

1. To view the file system for the failed container, run the **podman mount** command. For example, to view the file system for a failed **keystone** container, run the following command:

   ```
   $ sudo podman mount keystone
   ```

   This provides a mounted location to view the filesystem contents:

   ```
   /var/lib/containers/storage/overlay/78946a109085aeb8b3a350fc20bd8049a08918d74f573396d
   7358270e711c610/merged
   ```

   This is useful for viewing the Puppet reports within the container. You can find these reports in the **var/lib/puppet/** directory within the container mount.

Exporting a container

When a container fails, you might need to investigate the full contents of the file. In this case, you can export the full file system of a container as a **tar** archive. For example, to export the **keystone** container file system, run the following command:

```
$ sudo podman export keystone -o keystone.tar
```

This command creates the **keystone.tar** archive, which you can extract and explore.

14.8. TROUBLESHOOTING COMPUTE NODE FAILURES

Compute nodes use the Compute service to perform hypervisor-based operations. This means the main diagnosis for Compute nodes revolves around this service.

Procedure

1. Source the **stackrc** file:

   ```
   $ source ~/stackrc
   ```

2. Get the IP address of the Compute node that contains the failure:

   ```
   (undercloud) $ openstack server list
   ```

3. Log in to the node:

   ```
   (undercloud) $ ssh tripleo-admin@192.168.24.60
   ```

4. Change to the root user:

   ```
   $ sudo -i
   ```

5. View the status of the container:

   ```
   $ sudo podman ps -f name=nova_compute
   ```

6. The primary log file for Compute nodes is **/var/log/containers/nova/nova-compute.log**. If issues occur with Compute node communication, use this file to begin the diagnosis.

7. If you perform maintenance on the Compute node, migrate the existing instances from the host to an operational Compute node, then disable the node.

14.9. CREATING AN SOSREPORT

If you need to contact Red Hat for support with Red Hat OpenStack Platform, you might need to generate an **sosreport**. For more information about creating an **sosreport**, see:

- "How to collect all required logs for Red Hat Support to investigate an OpenStack issue"

14.10. LOG LOCATIONS

Use the following logs to gather information about the undercloud and overcloud when you troubleshoot issues.

Table 14.1. Logs on both the undercloud and overcloud nodes

Information	Log location
Containerized service logs	**/var/log/containers/**
Standard output from containerized services	**/var/log/containers/stdouts**
Ansible configuration logs	**~/ansible.log**

Table 14.2. Additional logs on the undercloud node

Information	Log location
Command history for **openstack overcloud deploy**	**/home/stack/.tripleo/history**
Undercloud installation log	**/home/stack/install-undercloud.log**

Table 14.3. Additional logs on the overcloud nodes

Information	Log location
Cloud-Init Log	**/var/log/cloud-init.log**
High availability log	**/var/log/pacemaker.log**

CHAPTER 15. TIPS FOR UNDERCLOUD AND OVERCLOUD SERVICES

This section provides advice on tuning and managing specific OpenStack services on the undercloud.

15.1. TUNING DEPLOYMENT PERFORMANCE

Red Hat OpenStack Platform director uses OpenStack Orchestration (heat) to conduct the main deployment and provisioning functions. Heat uses a series of workers to execute deployment tasks. To calculate the default number of workers, the director heat configuration halves the total CPU thread count of the undercloud. In this instance, thread count refers to the number of CPU cores multiplied by the hyper-threading value. For example, if your undercloud has a CPU with 16 threads, heat spawns 8 workers by default. The director configuration also uses a minimum and maximum cap by default:

Service	Minimum	Maximum
OpenStack Orchestration (heat)	4	24

However, you can set the number of workers manually with the **HeatWorkers** parameter in an environment file:

heat-workers.yaml

```
parameter_defaults:
  HeatWorkers: 16
```

undercloud.conf

```
custom_env_files: heat-workers.yaml
```

15.2. CHANGING THE SSL/TLS CIPHER RULES FOR HAPROXY

If you enabled SSL/TLS in the undercloud (see Section 4.2, "Undercloud configuration parameters"), you might want to harden the SSL/TLS ciphers and rules that are used with the HAProxy configuration. This hardening helps to avoid SSL/TLS vulnerabilities, such as the POODLE vulnerability.

Set the following hieradata using the **hieradata_override** undercloud configuration option:

tripleo::haproxy::ssl_cipher_suite

The cipher suite to use in HAProxy.

tripleo::haproxy::ssl_options

The SSL/TLS rules to use in HAProxy.

For example, you might want to use the following cipher and rules:

- Cipher: **ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA256:ECDHE-ECDSA-AES128-SHA:ECDHE-RSA-**

AES256-SHA384:ECDHE-RSA-AES128-SHA:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES256-SHA:ECDHE-RSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES128-SHA:DHE-RSA-AES256-SHA256:DHE-RSA-AES256-SHA:ECDHE-ECDSA-DES-CBC3-SHA:ECDHE-RSA-DES-CBC3-SHA:EDH-RSA-DES-CBC3-SHA:AES128-GCM-SHA256:AES256-GCM-SHA384:AES128-SHA256:AES256-SHA256:AES128-SHA:AES256-SHA:DES-CBC3-SHA:!DSS

- Rules: **no-sslv3 no-tls-tickets**

Create a hieradata override file (**haproxy-hiera-overrides.yaml**) with the following content:

```
tripleo::haproxy::ssl_cipher_suite: ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA256:ECDHE-ECDSA-AES128-SHA:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-AES128-SHA:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES256-SHA:ECDHE-RSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES128-SHA:DHE-RSA-AES256-SHA256:DHE-RSA-AES256-SHA:ECDHE-ECDSA-DES-CBC3-SHA:ECDHE-RSA-DES-CBC3-SHA:EDH-RSA-DES-CBC3-SHA:AES128-GCM-SHA256:AES256-GCM-SHA384:AES128-SHA256:AES256-SHA256:AES128-SHA:AES256-SHA:DES-CBC3-SHA:!DSS
tripleo::haproxy::ssl_options: no-sslv3 no-tls-tickets
```

NOTE

The cipher collection is one continuous line.

Set the **hieradata_override** parameter in the **undercloud.conf** file to use the hieradata override file you created before you ran **openstack undercloud install**:

```
[DEFAULT]
...
hieradata_override = haproxy-hiera-overrides.yaml
...
```

CHAPTER 16. POWER MANAGEMENT DRIVERS

Although IPMI is the main method that director uses for power management control, director also supports other power management types. This appendix contains a list of the power management features that director supports. Use these power management settings when you register nodes for the overcloud. For more information, see Registering nodes for the overcloud .

16.1. INTELLIGENT PLATFORM MANAGEMENT INTERFACE (IPMI)

The standard power management method when you use a baseboard management controller (BMC).

pm_type

Set this option to **ipmi**.

pm_user; pm_password

The IPMI username and password.

pm_addr

The IP address of the IPMI controller.

pm_port (Optional)

The port to connect to the IPMI controller.

16.2. REDFISH

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

pm_type

Set this option to **redfish**.

pm_user; pm_password

The Redfish username and password.

pm_addr

The IP address of the Redfish controller.

pm_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

If the Redfish service in your baseboard management controller (BMC) is not configured to use a valid TLS certificate signed by a recognized certificate authority (CA), the Redfish client in ironic fails to connect to the BMC. Set the **redfish_verify_ca** option to **false** to mute the error. However, be aware that disabling BMC authentication compromises the access security of your BMC.

16.3. DELL REMOTE ACCESS CONTROLLER (DRAC)

DRAC is an interface that provides out-of-band remote management features including power management and server monitoring.

pm_type

Set this option to **idrac**.

pm_user; pm_password

The DRAC username and password.

pm_addr

The IP address of the DRAC host.

16.4. INTEGRATED LIGHTS-OUT (ILO)

iLO from Hewlett-Packard is an interface that provides out-of-band remote management features including power management and server monitoring.

pm_type

Set this option to **ilo**.

pm_user; pm_password

The iLO username and password.

pm_addr

The IP address of the iLO interface.

- To enable this driver, add **ilo** to the **enabled_hardware_types** option in your **undercloud.conf** and rerun **openstack undercloud install**.

- HP nodes must have a minimum ILO firmware version of 1.85 (May 13 2015) for successful introspection. Director has been successfully tested with nodes using this ILO firmware version.

- Using a shared iLO port is not supported.

16.5. FUJITSU INTEGRATED REMOTE MANAGEMENT CONTROLLER (IRMC)

Fujitsu iRMC is a Baseboard Management Controller (BMC) with integrated LAN connection and extended functionality. This driver focuses on the power management for bare metal systems connected to the iRMC.

> **IMPORTANT**
>
> iRMC S4 or higher is required.

pm_type

Set this option to **irmc**.

pm_user; pm_password

The username and password for the iRMC interface.

> **IMPORTANT**
>
> The iRMC user must have the **ADMINISTRATOR** privilege.

pm_addr

The IP address of the iRMC interface.

pm_port (Optional)

The port to use for iRMC operations. The default is 443.

pm_auth_method (Optional)

The authentication method for iRMC operations. Use either **basic** or **digest**. The default is **basic**.

pm_client_timeout (Optional)

Timeout, in seconds, for iRMC operations. The default is 60 seconds.

pm_sensor_method (Optional)

Sensor data retrieval method. Use either **ipmitool** or **scci**. The default is **ipmitool**.

- To enable this driver, add **irmc** to the **enabled_hardware_types** option in your **undercloud.conf** and rerun the **openstack undercloud install** command.

16.6. MANUAL-MANAGEMENT DRIVER

Use the **manual-management** driver to control bare metal devices that do not have power management. Director does not control the registered bare metal devices, and you must perform manual power operations at certain points in the introspection and deployment processes.

> **IMPORTANT**
>
> This option is available only for testing and evaluation purposes. It is not recommended for Red Hat OpenStack Platform enterprise environments.

pm_type

Set this option to **manual-management**.

- This driver does not use any authentication details because it does not control power management.

- To enable this driver, add **manual-management** to the **enabled_hardware_types** option in your **undercloud.conf** and rerun the **openstack undercloud install** command.

- In your **instackenv.json** node inventory file, set the **pm_type** to **manual-management** for the nodes that you want to manage manually.

Introspection

- When performing introspection on nodes, manually start the nodes after running the **openstack overcloud node introspect** command. Ensure the nodes boot through PXE.

- If you have enabled node cleaning, manually reboot the nodes after the **Introspection completed** message appears and the node status is **clean wait** for each node when you run the **openstack baremetal node list** command. Ensure the nodes boot through PXE.

- After the introspection and cleaning process completes, shut down the nodes.

Deployment

- When performing overcloud deployment, check the node status with the **openstack baremetal node list** command. Wait until the node status changes from **deploying** to **wait call-back** and then manually start the nodes. Ensure the nodes boot through PXE.

- After the overcloud provisioning process completes, the nodes will shut down. You must boot the nodes from disk to start the configuration process. To check the completion of provisioning, check the node status with the **openstack baremetal node list** command, and wait until the node status changes to **active** for each node. When the node status is **active**, manually boot the provisioned overcloud nodes.