Red Hat OpenStack Platform 16.1

Director Installation and Usage

An end-to-end scenario on using Red Hat OpenStack Platform director to create an OpenStack cloud

Red Hat OpenStack Platform 16.1 Director Installation and Usage

An end-to-end scenario on using Red Hat OpenStack Platform director to create an OpenStack cloud

OpenStack Team
rhos-docs@redhat.com

Legal Notice

Abstract

Install Red Hat OpenStack Platform 16 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 .

CHAPTER 1. INTRODUCTION TO DIRECTOR

The 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 OpenStack nodes.

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

OPENSTACK_364004_0715
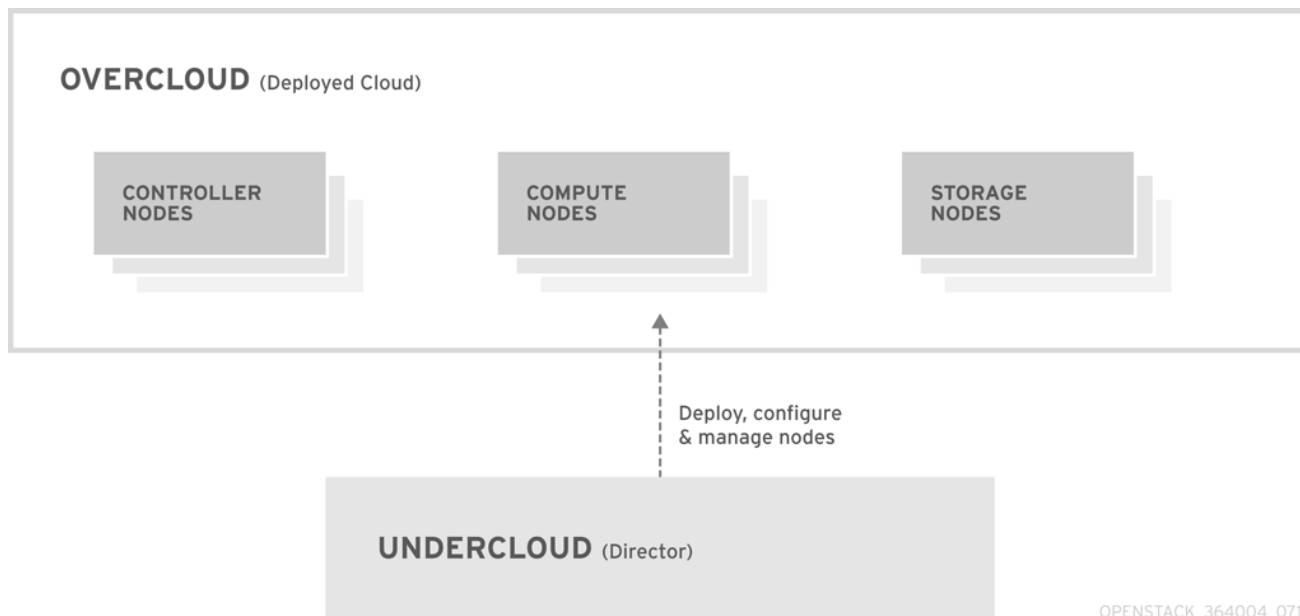
1.1. UNDERSTANDING THE UNDERCLOUD

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

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 Red Hat OpenStack Platform 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 OpenStack on each node. You can use this feature to provision bare metal systems as OpenStack nodes. For a full list of power management drivers, see Chapter 30, *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 OpenStack environment. The plans also include hooks that you can use to incorporate your own customizations as certain points in the environment creation process.

Undercloud components

The undercloud uses OpenStack components as its base tool set. Each component operates within a separate container on the undercloud:

- OpenStack Identity (keystone) – Provides authentication and authorization for the director components.

- OpenStack Bare Metal (ironic) and OpenStack Compute (nova) – Manages bare metal nodes.

- OpenStack Networking (neutron) and Open vSwitch – Control networking for bare metal nodes.

- OpenStack Image Service (glance) – Stores images that director writes to bare metal machines.

- OpenStack Orchestration (heat) and Puppet – Provides orchestration of nodes and configuration of nodes after director writes the overcloud image to disk.

- OpenStack Workflow Service (mistral) – Provides a set of workflows for certain director-specific actions, such as importing and deploying plans.

- OpenStack Messaging Service (zaqar) – Provides a messaging service for the OpenStack Workflow Service.

- OpenStack Object Storage (swift) – Provides object storage for various OpenStack Platform components, including:

  - Image storage for OpenStack Image Service

  - Introspection data for OpenStack Bare Metal

  - Deployment plans for OpenStack Workflow Service

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 OpenStack Platform 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 OpenStack environment. A recommended OpenStack 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:

- OpenStack Dashboard (horizon)

- OpenStack Identity (keystone)

- OpenStack Compute (nova) API

- OpenStack Networking (neutron)

- OpenStack Image Service (glance)

- OpenStack Block Storage (cinder)

- OpenStack Object Storage (swift)

- OpenStack Orchestration (heat)

- OpenStack Shared File Systems (manila)

- OpenStack Bare Metal (ironic)

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

- OpenStack Key Manager (barbican)

- MariaDB

- Open vSwitch

- Pacemaker and Galera for high availability services.

Compute

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

- OpenStack Compute (nova)

- KVM/QEMU

- OpenStack Telemetry (ceilometer) agent

- Open vSwitch

Storage

Storage nodes provide storage for the OpenStack 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:

  - OpenStack Block Storage (cinder) volume

  - OpenStack Telemetry agents

  - Open vSwitch.

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

  - OpenStack Object Storage (swift) storage

- OpenStack Telemetry agents

- Open vSwitch.

1.3. UNDERSTANDING HIGH AVAILABILITY IN RED HAT OPENSTACK PLATFORM

The Red Hat OpenStack Platform (RHOSP) director uses a Controller node cluster to provide highly available services to your OpenStack Platform environment. For each service, director installs the same components on all Controller nodes and manages the Controller nodes together as a single service. This type of cluster configuration provides a fallback in the event of operational failures on a single Controller node. This provides OpenStack users with a certain degree of continuous operation.

The OpenStack Platform director uses some key pieces of software to manage components on the Controller node:

- Pacemaker – Pacemaker is a cluster resource manager. Pacemaker manages and monitors the availability of OpenStack components across all nodes in the cluster.

- HAProxy – Provides load balancing and proxy services to the cluster.

- Galera – Replicates the RHOSP database across the cluster.

- Memcached – Provides database caching.

> **NOTE**
>
> - From version 13 and later, you can use director to deploy High Availability for Compute Instances (Instance HA). With Instance HA you can automate evacuating instances from a Compute node when the Compute node fails.

1.4. UNDERSTANDING CONTAINERIZATION IN RED HAT OPENSTACK PLATFORM

Each OpenStack Platform service on the undercloud and overcloud runs inside an individual Linux container on their respective node. This containerization provides a method to isolate services, maintain the environment, and upgrade Red Hat OpenStack Platform (RHOSP).

Red Hat OpenStack Platform 16.1 supports installation on the Red Hat Enterprise Linux 8.2 operating system. Red Hat Enterprise Linux 8.2 no longer includes Docker and provides a new set of tools to replace the Docker ecosystem. This means OpenStack Platform 16.1 replaces Docker with these new tools for OpenStack Platform deployment and upgrades.

Podman

Pod Manager (Podman) is a container management tool. It implements almost all Docker CLI commands, not including commands related to Docker Swarm. Podman manages pods, containers, and container images. One of the major differences between Podman and Docker is that Podman can manage resources without a daemon running in the background.
For more information about Podman, see the Podman website.

Buildah

Buildah specializes in building Open Containers Initiative (OCI) images, which you use in conjunction

with Podman. Buildah commands replicate the contents of a Dockerfile. Buildah also provides a lower-level **coreutils** interface to build container images, so that you do not require a Dockerfile to build containers. Buildah also uses other scripting languages to build container images without requiring a daemon.

For more information about Buildah, see the Buildah website.

Skopeo

Skopeo provides operators with a method to inspect remote container images, which helps director collect data when it pulls images. Additional features include copying container images from one registry to another and deleting images from registries.

Red Hat supports the following methods for managing container images for your overcloud:

- Pulling container images from the Red Hat Container Catalog to the **image-serve** registry on the undercloud and then pulling the images from the **image-serve** registry. When you pull images to the undercloud first, you avoid multiple overcloud nodes simultaneously pulling container images over an external connection.

- Pulling container images from your Satellite 6 server. You can pull these images directly from the Satellite because the network traffic is internal.

This guide contains information about configuring your container image registry details and performing basic container operations.

1.5. WORKING WITH CEPH STORAGE IN RED HAT OPENSTACK PLATFORM

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 glance (images), cinder (volumes), and nova (Compute) on a single node can become impossible to manage in large deployments with thousands of clients. Scaling OpenStack 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, Ceph 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 .

> **NOTE**
>
> For multi-architecture clouds, Red Hat supports only pre-installed or external Ceph implementation. For more information, see Integrating an Overcloud with an Existing Red Hat Ceph Cluster and Configuring the CPU architecture for the overcloud .

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. CONTAINERIZED UNDERCLOUD

The undercloud is the node that controls the configuration, installation, and management of your final Red Hat OpenStack Platform (RHOSP) environment, which is called the overcloud. The undercloud runs each RHOSP component service as a container. The undercloud uses these containerized services to create a toolset named director, which you use to create and manage your overcloud.

Since both the undercloud and overcloud use containers, both use the same architecture to pull, configure, and run containers. This architecture is based on the OpenStack Orchestration service (heat) for provisioning nodes and uses Ansible to configure services and containers. It is useful to have some familiarity with heat and Ansible to help you troubleshoot issues that you might encounter.

2.2. PREPARING YOUR UNDERCLOUD NETWORKING

The undercloud requires access to two main networks:

- The **Provisioning or Control Plane network**, which is 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. The director configures the interface for this network.

- The **External network**, which enables access to OpenStack Platform repositories, container image sources, and other servers such as DNS servers or NTP servers. Use this network for standard access the undercloud from your workstation. You must manually configure an interface on the undercloud to access the external network.

The undercloud requires a minimum of 2 x 1 Gbps Network Interface Cards: one for the **Provisioning or Control Plane network** and one for 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. Do not create provisioning and the control plane networks on top of an OVS bridge.

- 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.
    For best performance, Controllers with more than 50 compute nodes—or if more than four bare metal nodes are provisioned simultaneously—should have 4-10 times the bandwidth than 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.

2.3. 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 OpenStack 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
OpenStack Orchestration (heat)	4	24
All other service	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.

  - The **ceph-ansible** playbook consumes 1 GB resident set size (RSS) for every 10 hosts that the undercloud deploys. If you want to use a new or existing Ceph cluster in your deployment, you must provision the undercloud RAM accordingly.

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.

2.4. 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

2.5. VIRTUALIZATION 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 8, as listed on certified hypervisors.
Red Hat Virtualization	Hosted by Red Hat Virtualization 4.x, as listed on certified hypervisors.
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

Red Hat OpenStack Platform director requires that the latest version of Red Hat Enterprise Linux 8 is installed as the host operating system. This means your virtualization platform must also support the underlying Red Hat Enterprise Linux version.

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 to prevent the hypervisor blocking the undercloud from transmitting traffic from an unknown address. The configuration depends on the platform you are using for your virtual undercloud:

- Red Hat Enterprise Virtualization: Disable the **anti-mac-spoofing** parameter.

- VMware ESX or ESXi:

    - 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. Rebooting the VM is not sufficient.

2.6. CHARACTER ENCODING CONFIGURATION

Red Hat OpenStack Platform has special character encoding requirements as part of the locale settings:

- Use UTF-8 encoding on all nodes. Ensure the **LANG** environment variable is set to **en_US.UTF-8** on all nodes.

- Avoid using non-ASCII characters if you use Red Hat Ansible Tower to automate the creation of Red Hat OpenStack Platform resources.

2.7. CONSIDERATIONS WHEN RUNNING THE UNDERCLOUD WITH A PROXY

Running the undercloud with a proxy has certain limitations, and Red Hat recommends that you use Red Hat Satellite for registry and package management.

However, if your environment uses a proxy, review these considerations to best understand the different configuration methods of integrating parts of Red Hat OpenStack Platform with a proxy and the limitations of each method.

System-wide proxy configuration

Use this method to configure proxy communication for all network traffic on the undercloud. To configure the proxy settings, edit the **/etc/environment** file and set the following environment variables:

http_proxy

The proxy that you want to use for standard HTTP requests.

https_proxy

The proxy that you want to use for HTTPs requests.

no_proxy

A comma-separated list of domains that you want to exclude from proxy communications.

The system-wide proxy method has the following limitations:

- The **no_proxy** variable primarily uses domain names (**www.example.com**), domain suffixes (**example.com**), and domains with a wildcard (***.example.com**). Most Red Hat OpenStack Platform services interpret IP addresses in **no_proxy** but certain services, such as container health checks, do not interpret IP addresses in the **no_proxy** environment variable due to limitations with cURL and **wget**. To use a system-wide proxy with the undercloud, disable container health checks with the **container_healthcheck_disabled** parameter in the **undercloud.conf** file during installation. For more information, see BZ#1837458 – Container health checks fail to honor no_proxy CIDR notation.

- The maximum length of **no_proxy** is 1024 characters due to a fixed size buffer in the   **pam_env** PAM module.

- Some containers bind and parse the environment variables in **/etc/environments** incorrectly, which causes problems when running these services. For more information, see BZ#1916070 – proxy configuration updates in /etc/environment files are not being picked up in containers correctly and BZ#1918408 – mistral_executor container fails to properly set no_proxy environment parameter.

dnf proxy configuration

Use this method to configure **dnf** to run all traffic through a proxy. To configure the proxy settings, edit the **/etc/dnf/dnf.conf** file and set the following parameters:

proxy

The URL of the proxy server.

proxy_username

The username that you want to use to connect to the proxy server.

proxy_password

The password that you want to use to connect to the proxy server.

proxy_auth_method

The authentication method used by the proxy server.

For more information about these options, run **man dnf.conf**.

The **dnf** proxy method has the following limitations:

- This method provides proxy support only for **dnf**.

- The **dnf** proxy method does not include an option to exclude certain hosts from proxy communication.

Red Hat Subscription Manager proxy

Use this method to configure Red Hat Subscription Manager to run all traffic through a proxy. To configure the proxy settings, edit the **/etc/rhsm/rhsm.conf** file and set the following parameters:

proxy_hostname

Host for the proxy.

proxy_scheme

The scheme for the proxy when writing out the proxy to repo definitions.

proxy_port

The port for the proxy.

proxy_username

The username that you want to use to connect to the proxy server.

proxy_password

The password to use for connecting to the proxy server.

no_proxy

A comma-separated list of hostname suffixes for specific hosts that you want to exclude from proxy communication.

For more information about these options, run **man rhsm.conf**.

The Red Hat Subscription Manager proxy method has the following limitations:

- This method provides proxy support only for Red Hat Subscription Manager.

- The values for the Red Hat Subscription Manager proxy configuration override any values set for the system-wide environment variables.

Transparent proxy

If your network uses a transparent proxy to manage application layer traffic, you do not need to configure the undercloud itself to interact with the proxy because proxy management occurs automatically. A transparent proxy can help overcome limitations associated with client-based proxy configuration in Red Hat OpenStack Platform.

2.8. UNDERCLOUD REPOSITORIES

Red Hat OpenStack Platform (RHOSP) 16.1 runs on Red Hat Enterprise Linux 8.2. As a result, you must lock the content from these repositories to the respective Red Hat Enterprise Linux version.

NOTE

If you synchronize repositories by using Red Hat Satellite, you can enable specific versions of the Red Hat Enterprise Linux repositories. However, the repository label remains the same despite the version you choose. For example, if you enable the 8.2 version of the BaseOS repository, the repository name includes the specific version that you enabled, but the repository label is still **rhel-8-for-x86_64-baseos-tus-rpms**.

WARNING

Any repositories outside the ones specified here are not supported. Unless recommended, do not enable any other products or repositories outside 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 8.2 for x86_64 – BaseOS (RPMs) Telecommunications Update Service (TUS)	**rhel-8-for-x86_64-baseos-tus-rpms**	Base operating system repository for x86_64 systems.
Red Hat Enterprise Linux 8.2 for x86_64 – AppStream (RPMs)	**rhel-8-for-x86_64-appstream-tus-rpms**	Contains Red Hat OpenStack Platform dependencies.
Red Hat Enterprise Linux 8.2 for x86_64 – High Availability (RPMs) Telecommunications Update Service (TUS)	**rhel-8-for-x86_64-highavailability-tus-rpms**	High availability tools for Red Hat Enterprise Linux. Used for Controller node high availability.
Red Hat Ansible Engine 2.9 for RHEL 8 x86_64 (RPMs)	**ansible-2.9-for-rhel-8-x86_64-rpms**	Ansible Engine for Red Hat Enterprise Linux. Used to provide the latest version of Ansible.
Advanced Virtualization for RHEL 8 x86_64 (RPMs)	**advanced-virt-for-rhel-8-x86_64-eus-rpms**	Provides virtualization packages for OpenStack Platform.
Red Hat Satellite Tools for RHEL 8 Server RPMs x86_64	**satellite-tools-6.5-for-rhel-8-x86_64-rpms**	Tools for managing hosts with Red Hat Satellite 6.
Red Hat OpenStack Platform 16.1 for RHEL 8 (RPMs)	**openstack-16.1-for-rhel-8-x86_64-rpms**	Core Red Hat OpenStack Platform repository, which contains packages for Red Hat OpenStack Platform director.

Name	Repository	Description of requirement
Red Hat Fast Datapath for RHEL 8 (RPMS)	**fast-datapath-for-rhel-8-x86_64-rpms**	Provides Open vSwitch (OVS) packages for OpenStack Platform.

Ceph repositories

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

Name	Repository	Description of Requirement
Red Hat Ceph Storage Tools 4 for RHEL 8 x86_64 (RPMs)	**rhceph-4-tools-for-rhel-8-x86_64-rpms**	Provides tools for nodes to communicate with the Ceph Storage cluster. The undercloud requires the **ceph-ansible** package from this repository if you plan to use Ceph Storage in your overcloud or if you want to integrate with an existing Ceph Storage cluster.

IBM POWER repositories

The following table contains a list of repositories for RHOSP on POWER PC architecture. Use these repositories in place of equivalents in the Core repositories.

Name	Repository	Description of requirement
Red Hat Enterprise Linux for IBM Power, little endian – BaseOS (RPMs)	**rhel-8-for-ppc64le-baseos-rpms**	Base operating system repository for ppc64le systems.
Red Hat Enterprise Linux 8 for IBM Power, little endian – AppStream (RPMs)	**rhel-8-for-ppc64le-appstream-rpms**	Contains Red Hat OpenStack Platform dependencies.
Red Hat Enterprise Linux 8 for IBM Power, little endian – High Availability (RPMs)	**rhel-8-for-ppc64le-highavailability-rpms**	High availability tools for Red Hat Enterprise Linux. Used for Controller node high availability.
Red Hat Fast Datapath for RHEL 8 IBM Power, little endian (RPMS)	**fast-datapath-for-rhel-8-ppc64le-rpms**	Provides Open vSwitch (OVS) packages for OpenStack Platform.
Red Hat Ansible Engine 2.8 for RHEL 8 IBM Power, little endian (RPMs)	**ansible-2.8-for-rhel-8-ppc64le-rpms**	Ansible Engine for Red Hat Enterprise Linux. Provides the latest version of Ansible.

Name	Repository	Description of requirement
Red Hat OpenStack Platform 16.1 for RHEL 8 (RPMs)	**openstack-16.1-for-rhel-8-ppc64le-rpms**	Core Red Hat OpenStack Platform repository for ppc64le systems.

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
   [stack@director ~]$ sudo hostnamectl set-hostname --transient 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

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)
                        Red Hat Virtualization
   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 16.1 entitlement:

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

5. Lock the undercloud to Red Hat Enterprise Linux 8.2:

```
$ sudo subscription-manager release --set=8.2
```

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 repositories:

```
[stack@director ~]$ sudo subscription-manager repos --disable=*
[stack@director ~]$ sudo subscription-manager repos --enable=rhel-8-for-x86_64-baseos-
tus-rpms --enable=rhel-8-for-x86_64-appstream-tus-rpms --enable=rhel-8-for-x86_64-
highavailability-tus-rpms --enable=ansible-2.9-for-rhel-8-x86_64-rpms --enable=openstack-
16.1-for-rhel-8-x86_64-rpms --enable=fast-datapath-for-rhel-8-x86_64-rpms --
enable=advanced-virt-for-rhel-8-x86_64-eus-rpms
```

These repositories contain packages that the director installation requires.

3. Set the **container-tools** repository module to version **2.0**:

```
[stack@director ~]$ sudo dnf module reset container-tools
[stack@director ~]$ sudo dnf module enable -y container-tools:2.0
```

4. 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
```

3.4. INSTALLING DIRECTOR PACKAGES

Install packages relevant to Red Hat OpenStack Platform director.

Procedure

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

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

3.5. INSTALLING CEPH-ANSIBLE

The **ceph-ansible** package is required when you use Ceph Storage with Red Hat OpenStack Platform.

Procedure

1. Enable the Ceph Tools repository:

   ```
   [stack@director ~]$ sudo subscription-manager repos --enable=rhceph-4-tools-for-rhel-8-x86_64-rpms
   ```

2. Install the **ceph-ansible** package:

   ```
   [stack@director ~]$ sudo dnf install -y ceph-ansible
   ```

3.6. 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 this 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 your undercloud host as the **stack** user.

2. Generate the default container image preparation file:

   ```
   $ sudo 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.

3.7. CONTAINER IMAGE PREPARATION PARAMETERS

The default file for preparing your containers (**containers-prepare-parameter.yaml**) contains the **ContainerImagePrepare** heat parameter. This parameter defines a list of strategies for preparing a set of images:

```
parameter_defaults:
  ContainerImagePrepare:
  - (strategy one)
  - (strategy two)
  - (strategy three)
  ...
```

Each strategy accepts a set of sub-parameters that defines which images to use and what to do with the images. The following table contains information about the sub-parameters that you can use with each **ContainerImagePrepare** strategy:

Parameter	Description
excludes	List of regular expressions to exclude image names from a strategy.
includes	List of regular expressions to include in a strategy. At least one image name must match an existing image. All **excludes** are ignored if **includes** is specified.
modify_append_tag	String to append to the tag for the destination image. For example, if you pull an image with the tag 16.1.3-5.161 and set the **modify_append_tag** to **-hotfix**, the director tags the final image as 16.1.3-5.161-hotfix.
modify_only_with_labels	A dictionary of image labels that filter the images that you want to modify. If an image matches the labels defined, the director includes the image in the modification process.
modify_role	String of ansible role names to run during upload but before pushing the image to the destination registry.
modify_vars	Dictionary of variables to pass to **modify_role**.

Parameter	Description
push_destination	Defines the namespace of the registry that you want to push images to during the upload process. • If set to **true**, the **push_destination** is set to the undercloud registry namespace using the hostname, which is the recommended method. • If set to **false**, the push to a local registry does not occur and nodes pull images directly from the source. • If set to a custom value, director pushes images to an external local registry. If you set this parameter to **false** in production environments while pulling images directly from Red Hat Container Catalog, all overcloud nodes will simultaneously pull the images from the Red Hat Container Catalog over your external connection, which can cause bandwidth issues. Only use **false** to pull directly from a Red Hat Satellite Server hosting the container images. If the **push_destination** parameter is set to **false** or is not defined and the remote registry requires authentication, set the **ContainerImageRegistryLogin** parameter to **true** and include the credentials with the **ContainerImageRegistryCredentials** parameter.
pull_source	The source registry from where to pull the original container images.
set	A dictionary of **key: value** definitions that define where to obtain the initial images.
tag_from_label	Use the value of specified container image metadata labels to create a tag for every image and pull that tagged image. For example, if you set **tag_from_label: {version}-{release}**, director uses the **version** and **release** labels to construct a new tag. For one container, **version** might be set to 16.1.3 and **release** might be set to **5.161**, which results in the tag 16.1.3-5.161. Director uses this parameter only if you have not defined **tag** in the **set** dictionary.

IMPORTANT

When you push images to the undercloud, use **push_destination: true** instead of **push_destination: UNDERCLOUD_IP:PORT**. The **push_destination: true** method provides a level of consistency across both IPv4 and IPv6 addresses.

The **set** parameter accepts a set of **key: value** definitions:

Key	Description
ceph_image	The name of the Ceph Storage container image.
ceph_namespace	The namespace of the Ceph Storage container image.
ceph_tag	The tag of the Ceph Storage container image.
ceph_alertmanager_image **ceph_alertmanager_namespace** **ceph_alertmanager_tag**	The name, namespace, and tag of the Ceph Storage Alert Manager container image.
ceph_grafana_image **ceph_grafana_namespace** **ceph_grafana_tag**	The name, namespace, and tag of the Ceph Storage Grafana container image.
ceph_node_exporter_image **ceph_node_exporter_namespace** **ceph_node_exporter_tag**	The name, namespace, and tag of the Ceph Storage Node Exporter container image.
ceph_prometheus_image **ceph_prometheus_namespace** **ceph_prometheus_tag**	The name, namespace, and tag of the Ceph Storage Prometheus container image.
name_prefix	A prefix for each OpenStack service image.
name_suffix	A suffix for each OpenStack service image.
namespace	The namespace for each OpenStack service image.
neutron_driver	The driver to use to determine which OpenStack Networking (neutron) container to use. Use a null value to set to the standard **neutron-server** container. Set to **ovn** to use OVN-based containers.

Key	Description
tag	Sets a specific tag for all images from the source. If not defined, director uses the Red Hat OpenStack Platform version number as the default value. This parameter takes precedence over the **tag_from_label** value.

> **NOTE**
>
> The container images use multi-stream tags based on the Red Hat OpenStack Platform version. This means that there is no longer a **latest** tag.

3.8. GUIDELINES FOR CONTAINER IMAGE TAGGING

The Red Hat Container Registry uses a specific version format to tag all Red Hat OpenStack Platform container images. This format follows the label metadata for each container, which is **version-release**.

version

Corresponds to a major and minor version of Red Hat OpenStack Platform. These versions act as streams that contain one or more releases.

release

Corresponds to a release of a specific container image version within a version stream.

For example, if the latest version of Red Hat OpenStack Platform is 16.1.3 and the release for the container image is **5.161**, then the resulting tag for the container image is 16.1.3-5.161.

The Red Hat Container Registry also uses a set of major and minor **version** tags that link to the latest release for that container image version. For example, both 16.1 and 16.1.3 link to the latest **release** in the 16.1.3 container stream. If a new minor release of 16.1 occurs, the 16.1 tag links to the latest **release** for the new minor release stream while the 16.1.3 tag continues to link to the latest **release** within the 16.1.3 stream.

The **ContainerImagePrepare** parameter contains two sub-parameters that you can use to determine which container image to download. These sub-parameters are the **tag** parameter within the **set** dictionary, and the **tag_from_label** parameter. Use the following guidelines to determine whether to use **tag** or **tag_from_label**.

- The default value for **tag** is the major version for your OpenStack Platform version. For this version it is 16.1. This always corresponds to the latest minor version and release.

  ```
  parameter_defaults:
    ContainerImagePrepare:
    - set:
        ...
        tag: 16.1
        ...
  ```

- To change to a specific minor version for OpenStack Platform container images, set the tag to a minor version. For example, to change to 16.1.2, set **tag** to 16.1.2.

  ```
  parameter_defaults:
  ```

```
ContainerImagePrepare:
- set:
    ...
    tag: 16.1.2
    ...
```

- When you set **tag**, director always downloads the latest container image **release** for the version set in **tag** during installation and updates.

- If you do not set **tag**, director uses the value of **tag_from_label** in conjunction with the latest major version.

```
parameter_defaults:
  ContainerImagePrepare:
  - set:
      ...
      # tag: 16.1
      ...
    tag_from_label: '{version}-{release}'
```

- The **tag_from_label** parameter generates the tag from the label metadata of the latest container image release it inspects from the Red Hat Container Registry. For example, the labels for a certain container might use the following **version** and **release** metadata:

```
"Labels": {
  "release": "5.161",
  "version": "16.1.3",
  ...
}
```

- The default value for **tag_from_label** is **{version}-{release}**, which corresponds to the version and release metadata labels for each container image. For example, if a container image has 16.1.3 set for **version** and 5.161 set for **release**, the resulting tag for the container image is 16.1.3-5.161.

- The **tag** parameter always takes precedence over the **tag_from_label** parameter. To use **tag_from_label**, omit the **tag** parameter from your container preparation configuration.

- A key difference between **tag** and **tag_from_label** is that director uses **tag** to pull an image only based on major or minor version tags, which the Red Hat Container Registry links to the latest image release within a version stream, while director uses **tag_from_label** to perform a metadata inspection of each container image so that director generates a tag and pulls the corresponding image.

3.9. 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
```

3.10. LAYERING IMAGE PREPARATION ENTRIES

The value of the **ContainerImagePrepare** parameter is a YAML list. This means that you can specify
multiple entries. The following example demonstrates two entries where director uses the latest version
of all images except for the **nova-api** image, which uses the version tagged with **16.2-44**:

```
ContainerImagePrepare:
- tag_from_label: "{version}-{release}"
  push_destination: true
  excludes:
  - nova-api
  set:
    namespace: registry.redhat.io/rhosp-rhel8
    name_prefix: openstack-
    name_suffix: ''
- push_destination: true
  includes:
  - nova-api
  set:
    namespace: registry.redhat.io/rhosp-rhel8
    tag: 16.2-44
```

The **includes** and **excludes** parameters use regular expressions to control image filtering for each entry. The images that match the **includes** strategy take precedence over **excludes** matches. The image name must the **includes** or **excludes** regular expression value to be considered a match.

3.11. EXCLUDING CEPH STORAGE CONTAINER IMAGES

The default overcloud role configuration uses the default Controller, Compute, and Ceph Storage roles. However, if you use the default role configuration to deploy an overcloud without Ceph Storage nodes, director still pulls the Ceph Storage container images from the Red Hat Container Registry because the images are included as a part of the default configuration.

If your overcloud does not require Ceph Storage containers, you can configure director to not pull the Ceph Storage containers images from the Red Hat Container Registry.

Procedure

1. Edit the **containers-prepare-parameter.yaml** file to exclude the Ceph Storage containers:

   ```
   parameter_defaults:
     ContainerImagePrepare:
     - push_destination: true
       excludes:
         - ceph
         - prometheus
       set:
         …
   ```

   The **excludes** parameter uses regular expressions to exclude any container images that contain the **ceph** or **prometheus** strings.

2. Save the **containers-prepare-parameter.yaml** file.

3.12. MODIFYING IMAGES DURING PREPARATION

It is possible to modify images during image preparation, and then immediately deploy the overcloud with modified images.

> **NOTE**
>
> Red Hat OpenStack Platform (RHOSP) director supports modifying images during preparation for RHOSP containers, not for Ceph containers.

Scenarios for modifying images include:

- As part of a continuous integration pipeline where images are modified with the changes being tested before deployment.

- As part of a development workflow where local changes must be deployed for testing and development.

- When changes must be deployed but are not available through an image build pipeline. For example, adding proprietary add-ons or emergency fixes.

To modify an image during preparation, invoke an Ansible role on each image that you want to modify.

The role takes a source image, makes the requested changes, and tags the result. The prepare command can push the image to the destination registry and set the heat parameters to refer to the modified image.

The Ansible role **tripleo-modify-image** conforms with the required role interface and provides the behaviour necessary for the modify use cases. Control the modification with the modify-specific keys in the **ContainerImagePrepare** parameter:

- **modify_role** specifies the Ansible role to invoke for each image to modify.

- **modify_append_tag** appends a string to the end of the source image tag. This makes it obvious that the resulting image has been modified. Use this parameter to skip modification if the **push_destination** registry already contains the modified image. Change **modify_append_tag** whenever you modify the image.

- **modify_vars** is a dictionary of Ansible variables to pass to the role.

To select a use case that the **tripleo-modify-image** role handles, set the **tasks_from** variable to the required file in that role.

While developing and testing the **ContainerImagePrepare** entries that modify images, run the image prepare command without any additional options to confirm that the image is modified as you expect:

```
sudo openstack tripleo container image prepare \
  -e ~/containers-prepare-parameter.yaml
```

> **IMPORTANT**
>
> To use the **openstack tripleo container image prepare** command, your undercloud must contain a running **image-serve** registry. As a result, you cannot run this command before a new undercloud installation because the **image-serve** registry will not be installed. You can run this command after a successful undercloud installation.

3.13. UPDATING EXISTING PACKAGES ON CONTAINER IMAGES

> **NOTE**
>
> Red Hat OpenStack Platform (RHOSP) director supports updating existing packages on container images for RHOSP containers, not for Ceph containers.

Procedure

- The following example **ContainerImagePrepare** entry updates in all packages on the container images by using the dnf repository configuration of the undercloud host:

```
ContainerImagePrepare:
- push_destination: true
  ...
  modify_role: tripleo-modify-image
  modify_append_tag: "-updated"
  modify_vars:
    tasks_from: yum_update.yml
```

```
compare_host_packages: true
yum_repos_dir_path: /etc/yum.repos.d
...
```

3.14. INSTALLING ADDITIONAL RPM FILES TO CONTAINER IMAGES

You can install a directory of RPM files in your container images. This is useful for installing hotfixes, local package builds, or any package that is not available through a package repository.

NOTE

Red Hat OpenStack Platform (RHOSP) director supports installing additional RPM files to container images for RHOSP containers, not for Ceph containers.

Procedure

- The following example **ContainerImagePrepare** entry installs some hotfix packages on only the **nova-compute** image:

```
ContainerImagePrepare:
- push_destination: true
  ...
  includes:
  - nova-compute
  modify_role: tripleo-modify-image
  modify_append_tag: "-hotfix"
  modify_vars:
    tasks_from: rpm_install.yml
    rpms_path: /home/stack/nova-hotfix-pkgs
  ...
```

3.15. MODIFYING CONTAINER IMAGES WITH A CUSTOM DOCKERFILE

You can specify a directory that contains a Dockerfile to make the required changes. When you invoke the **tripleo-modify-image** role, the role generates a **Dockerfile.modified** file that changes the **FROM** directive and adds extra **LABEL** directives.

NOTE

Red Hat OpenStack Platform (RHOSP) director supports modifying container images with a custom Dockerfile for RHOSP containers, not for Ceph containers.

Procedure

1. The following example runs the custom Dockerfile on the **nova-compute** image:

```
ContainerImagePrepare:
- push_destination: true
  ...
  includes:
  - nova-compute
  modify_role: tripleo-modify-image
  modify_append_tag: "-hotfix"
```

```
modify_vars:
  tasks_from: modify_image.yml
  modify_dir_path: /home/stack/nova-custom
...
```

2. The following example shows the **/home/stack/nova-custom/Dockerfile** file. After you run any **USER** root directives, you must switch back to the original image default user:

```
FROM registry.redhat.io/rhosp-rhel8/openstack-nova-compute:latest

USER "root"

COPY customize.sh /tmp/
RUN /tmp/customize.sh

USER "nova"
```

3.16. PREPARING A SATELLITE SERVER FOR CONTAINER IMAGES

Red Hat Satellite 6 offers registry synchronization capabilities. This provides a method to pull multiple images into a Satellite server and manage them as part of an application life cycle. The Satellite also acts as a registry for other container-enabled systems to use. For more information about managing container images, see Managing Container Images in the *Red Hat Satellite 6 Content Management Guide*.

The examples in this procedure use the **hammer** command line tool for Red Hat Satellite 6 and an example organization called **ACME**. Substitute this organization for your own Satellite 6 organization.

> **NOTE**
>
> This procedure requires authentication credentials to access container images from **registry.redhat.io**. 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. For more information, see "Red Hat Container Registry Authentication".

Procedure

1. Create a list of all container images:

```
$ sudo podman search --limit 1000 "registry.redhat.io/rhosp-rhel8/openstack" --format="{{
.Name }}" | sort > satellite_images
$ sudo podman search --limit 1000 "registry.redhat.io/rhceph" | grep rhceph-4-dashboard-
rhel8
$ sudo podman search --limit 1000 "registry.redhat.io/rhceph" | grep rhceph-4-rhel8
$ sudo podman search --limit 1000 "registry.redhat.io/openshift" | grep ose-prometheus
```

- If you plan to install Ceph and enable the Ceph Dashboard, you need the following ose-prometheus containers:

```
registry.redhat.io/openshift4/ose-prometheus-node-exporter:v4.6
registry.redhat.io/openshift4/ose-prometheus:v4.6
registry.redhat.io/openshift4/ose-prometheus-alertmanager:v4.6
```

2. Copy the **satellite_images** file to a system that contains the Satellite 6 **hammer** tool. Alternatively, use the instructions in the *Hammer CLI Guide* to install the **hammer** tool to the undercloud.

3. Run the following **hammer** command to create a new product (**OSP16.1 Containers**) in your Satellite organization:

```
$ hammer product create \
  --organization "ACME" \
  --name "OSP Containers"
```

This custom product will contain your images.

4. Add the overcloud container images from the **satellite_images** file:

```
$ while read IMAGE; do \
  IMAGE_NAME=$(echo $IMAGE | cut -d"/" -f3 | sed "s/openstack-//g") ; \
  IMAGE_NOURL=$(echo $IMAGE | sed "s/registry.redhat.io\///g") ; \
  hammer repository create \
  --organization "ACME" \
  --product "OSP Containers" \
  --content-type docker \
  --url https://registry.redhat.io \
  --docker-upstream-name $IMAGE_NOURL \
  --upstream-username USERNAME \
  --upstream-password PASSWORD \
  --name $IMAGE_NAME ; done < satellite_images
```

5. Add the Ceph Storage 4 container image:

```
$ hammer repository create \
  --organization "ACME" \
  --product "OSP Containers" \
  --content-type docker \
  --url https://registry.redhat.io \
  --docker-upstream-name rhceph/rhceph-4-rhel8 \
  --upstream-username USERNAME \
  --upstream-password PASSWORD \
  --name rhceph-4-rhel8
```

> **NOTE**
>
> If you want to install the Ceph dashboard, include **--name rhceph-4-dashboard-rhel8** in the **hammer repository create** command:
>
> ```
> $ hammer repository create \
>   --organization "ACME" \
>   --product "OSP Containers" \
>   --content-type docker \
>   --url https://registry.redhat.io \
>   --docker-upstream-name rhceph/rhceph-4-dashboard-rhel8 \
>   --upstream-username USERNAME \
>   --upstream-password PASSWORD \
>   --name rhceph-4-dashboard-rhel8
> ```

6. Synchronize the container images:

```
$ hammer product synchronize \
  --organization "ACME" \
  --name "OSP Containers"
```

Wait for the Satellite server to complete synchronization.

> **NOTE**
>
> Depending on your configuration, **hammer** might ask for your Satellite server username and password. You can configure **hammer** to automatically login using a configuration file. For more information, see the Authentication section in the *Hammer CLI Guide*.

7. If your Satellite 6 server uses content views, create a new content view version to incorporate the images and promote it along environments in your application life cycle. This largely depends on how you structure your application lifecycle. For example, if you have an environment called **production** in your lifecycle and you want the container images to be available in that environment, create a content view that includes the container images and promote that content view to the **production** environment. For more information, see Managing Content Views.

8. Check the available tags for the **base** image:

```
$ hammer docker tag list --repository "base" \
  --organization "ACME" \
  --lifecycle-environment "production" \
  --product "OSP Containers"
```

This command displays tags for the OpenStack Platform container images within a content view for a particular environment.

9. Return to the undercloud and generate a default environment file that prepares images using your Satellite server as a source. Run the following example command to generate the environment file:

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

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

10. Edit the **containers-prepare-parameter.yaml** file and modify the following parameters:

- **push_destination** - Set this to **true** or **false** depending on your chosen container image management strategy. If you set this parameter to **false**, the overcloud nodes pull images directly from the Satellite. If you set this parameter to **true**, the director pulls the images from the Satellite to the undercloud registry and the overcloud pulls the images from the undercloud registry.

- **namespace** – The URL and port of the registry on the Satellite server. The default registry port on Red Hat Satellite is 443.

- **name_prefix** – The prefix is based on a Satellite 6 convention. This differs depending on whether you use content views:

  - If you use content views, the structure is **[org]-[environment]-[content view]-[product]-**. For example: **acme-production-myosp16-osp_containers-**.

  - If you do not use content views, the structure is **[org]-[product]-**. For example: **acme-osp_containers-**.

- **ceph_namespace**, **ceph_image**, **ceph_tag** – If you use Ceph Storage, include these additional parameters to define the Ceph Storage container image location. Note that **ceph_image** now includes a Satellite-specific prefix. This prefix is the same value as the **name_prefix** option.

The following example environment file contains Satellite-specific parameters:

```
parameter_defaults:
  ContainerImagePrepare:
  - push_destination: false
    set:
      ceph_image: acme-production-myosp16_1-osp_containers-rhceph-4
      ceph_namespace: satellite.example.com:443
      ceph_tag: latest
      name_prefix: acme-production-myosp16_1-osp_containers-
      name_suffix: ''
      namespace: satellite.example.com:443
      neutron_driver: null
      tag: '{osp_curr_ver_no_beta}'
      ...
```
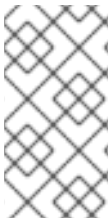
NOTE

To use a specific container image version stored on your Red Hat Satellite Server, set the **tag** key-value pair to the specific version in the **set** dictionary. For example, to use the {osp_curr_ver_no_beta}.2 image stream, set **tag: {osp_curr_ver_no_beta}.2** in the **set** dictionary.

You must define the **containers-prepare-parameter.yaml** environment file in the **undercloud.conf** configuration file, otherwise the undercloud uses the default values:

```
container_images_file = /home/stack/containers-prepare-parameter.yaml
```

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 DIRECTOR

The director installation process requires certain settings in the **undercloud.conf** configuration file, which director reads from the home directory of the **stack** user. Complete the following steps to copy default template as a foundation for your configuration.

Procedure

1. Copy the default template to the home directory of the **stack** user's:

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

2. Edit the **undercloud.conf** file. This file contains settings to configure your undercloud. If you omit or comment out a parameter, the undercloud installation uses the default value.

4.2. DIRECTOR 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 **ppc64le** architecture in addition to the default **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 8.2 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_mistral; enable_nova; enable_tempest; enable_validations; enable_zaqar

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_novajoin

Defines whether to install the **novajoin** metadata service in the undercloud.

enable_routed_networks

Defines whether to enable support for routed control plane networks.

enable_swift_encryption

Defines whether to enable Swift encryption at-rest.

enable_telemetry

Defines whether to install OpenStack Telemetry services (gnocchi, aodh, panko) in the undercloud. Set the **enable_telemetry** parameter to **true** if you want to install and configure telemetry services

automatically. The default value is **false**, which disables telemetry on the undercloud. This parameter is required if you use other products that consume metrics data, such as Red Hat CloudForms.

> **WARNING**
>
> RBAC is not supported by every component. The Alarming service (aodh) and Gnocchi do not take secure RBAC rules into account.

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.

ipa_otp

Defines the one-time password to register the undercloud node to an IPA server. This is required when **enable_novajoin** is enabled.

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.

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 set the **ControlVirtualInterface** parameter to 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. Set the **ControlVirtualInterface** 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_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.
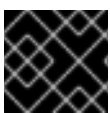
IMPORTANT

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 your nodes.

dhcp_exclude

IP addresses to exclude in the DHCP allocation range.

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.

Modify the values of these parameters to suit your configuration. When complete, save the file.

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
   ```

   Save this file when you have finished.

3. 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
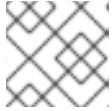
> **NOTE**
>
> You can specify multiple environment files using a comma-separated list.

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. CONFIGURING THE UNDERCLOUD FOR BARE METAL PROVISIONING OVER IPV6

If you have IPv6 nodes and infrastructure, you can configure the undercloud and the provisioning network to use IPv6 instead of IPv4 so that director can provision and deploy Red Hat OpenStack Platform onto IPv6 nodes. However, there are some considerations:

- Dual stack IPv4/6 is not available.

- Tempest validations might not perform correctly.

- IPv4 to IPv6 migration is not available during upgrades.

Modify the **undercloud.conf** file to enable IPv6 provisioning in Red Hat OpenStack Platform.

Prerequisites

- An IPv6 address on the undercloud. For more information, see Configuring an IPv6 address on the undercloud in the *IPv6 Networking for the Overcloud* guide.

Procedure

1. Open your **undercloud.conf** file.

2. Specify the IPv6 address mode as either stateless or stateful:

```
[DEFAULT]
ipv6_address_mode = <address_mode>
...
```

- Replace **<address_mode>** with **dhcpv6-stateless** or **dhcpv6-stateful**, based on the mode that your NIC supports.

> **NOTE**
>
> When you use the stateful address mode, the firmware, chain loaders, and operating systems might use different algorithms to generate an ID that the DHCP server tracks. DHCPv6 does not track addresses by MAC, and does not provide the same address back if the identifier value from the requester changes but the MAC address remains the same. Therefore, when you use stateful DHCPv6 you must also complete the next step to configure the network interface.

3. If you configured your undercloud to use stateful DHCPv6, specify the network interface to use for bare metal nodes:

   ```
   [DEFAULT]
   ipv6_address_mode = dhcpv6-stateful
   ironic_enabled_network_interfaces = neutron,flat
   ...
   ```

4. Set the default network interface for bare metal nodes:

   ```
   [DEFAULT]
   ...
   ironic_default_network_interface = neutron
   ...
   ```

5. Specify whether or not the undercloud should create a router on the provisioning network:

   ```
   [DEFAULT]
   ...
   enable_routed_networks: <true/false>
   ...
   ```

   - Replace **<true/false>** with **true** to enable routed networks and prevent the undercloud creating a router on the provisioning network. When **true**, the data center router must provide router advertisements.

   - Replace **<true/false>** with **false** to disable routed networks and create a router on the provisioning network.

6. Configure the local IP address, and the IP address for the director Admin API and Public API endpoints over SSL/TLS:

   ```
   [DEFAULT]
   ...
   local_ip = <ipv6_address>
   undercloud_admin_host = <ipv6_address>
   undercloud_public_host = <ipv6_address>
   ...
   ```

   - Replace **<ipv6_address>** with the IPv6 address of the undercloud.

7. Optional: Configure the provisioning network that director uses to manage instances:

```
[ctlplane-subnet]
cidr = <ipv6_address>/<ipv6_prefix>
...
```

- Replace **<ipv6_address>** with the IPv6 address of the network to use for managing instances when not using the default provisioning network.

- Replace **<ipv6_prefix>** with the IP address prefix of the network to use for managing instances when not using the default provisioning network.

8. Configure the DHCP allocation range for provisioning nodes:

```
[ctlplane-subnet]
cidr = <ipv6_address>/<ipv6_prefix>
dhcp_start = <ipv6_address_dhcp_start>
dhcp_end = <ipv6_address_dhcp_end>
...
```

- Replace **<ipv6_address_dhcp_start>** with the IPv6 address of the start of the network range to use for the overcloud nodes.

- Replace **<ipv6_address_dhcp_end>** with the IPv6 address of the end of the network range to use for the overcloud nodes.

9. Optional: Configure the gateway for forwarding traffic to the external network:

```
[ctlplane-subnet]
cidr = <ipv6_address>/<ipv6_prefix>
dhcp_start = <ipv6_address_dhcp_start>
dhcp_end = <ipv6_address_dhcp_end>
gateway = <ipv6_gateway_address>
...
```

- Replace **<ipv6_gateway_address>** with the IPv6 address of the gateway when not using the default gateway.

10. Configure the DHCP range to use during the inspection process:

```
[ctlplane-subnet]
cidr = <ipv6_address>/<ipv6_prefix>
dhcp_start = <ipv6_address_dhcp_start>
dhcp_end = <ipv6_address_dhcp_end>
gateway = <ipv6_gateway_address>
inspection_iprange = <ipv6_address_inspection_start>,<ipv6_address_inspection_end>
...
```

- Replace **<ipv6_address_inspection_start>** with the IPv6 address of the start of the network range to use during the inspection process.

- Replace **<ipv6_address_inspection_end>** with the IPv6 address of the end of the network range to use during the inspection process.

> **NOTE**
>
> This range must not overlap with the range defined by **dhcp_start** and **dhcp_end**, but must be in the same IP subnet.

11. Configure an IPv6 nameserver for the subnet:

```
[ctlplane-subnet]
cidr = <ipv6_address>/<ipv6_prefix>
dhcp_start = <ipv6_address_dhcp_start>
dhcp_end = <ipv6_address_dhcp_end>
gateway = <ipv6_gateway_address>
inspection_iprange = <ipv6_address_inspection_start>,<ipv6_address_inspection_end>
dns_nameservers = <ipv6_dns>
```

- Replace **<ipv6_dns>** with the DNS nameservers specific to the subnet.

4.7. CONFIGURING UNDERCLOUD NETWORK INTERFACES

Include custom network configuration in the **undercloud.conf** file to install the undercloud with specific networking functionality. For example, some interfaces might not have DHCP. In this case, you must disable DHCP for these interfaces in the **undercloud.conf** file so that **os-net-config** can apply the configuration during the undercloud installation process.

Procedure

1. Log in to the undercloud host.

2. Create a new file **undercloud-os-net-config.yaml** and include the network configuration that you require.
   For more information, see Network interface reference.

   Here is an example:

   ```
   network_config:
   - name: br-ctlplane
     type: ovs_bridge
     use_dhcp: false
     dns_servers: 192.168.122.1
     domain: lab.example.com
     ovs_extra:
     - "br-set-external-id br-ctlplane bridge-id br-ctlplane"
     addresses:
     - ip_netmask: 172.20.0.1/26
     members:
     - type: interface
       name: nic2
   ```

   To create a network bond for a specific interface, use the following sample:

   ```
   network_config:
   - name: br-ctlplane
     type: ovs_bridge
     use_dhcp: false
   ```

```
dns_servers: 192.168.122.1
domain: lab.example.com
ovs_extra:
- "br-set-external-id br-ctlplane bridge-id br-ctlplane"
addresses:
- ip_netmask: 172.20.0.1/26
members:
- name: bond-ctlplane
  type: linux_bond
  use_dhcp: false
  bonding_options: "mode=active-backup"
  mtu: 1500
  members:
  - type: interface
    name: nic2
  - type: interface
    name: nic3
```

3. Include the path to the **undercloud-os-net-config.yaml** file in the **net_config_override** parameter in the **undercloud.conf** file:

```
[DEFAULT]
...
net_config_override=undercloud-os-net-config.yaml
...
```

> **NOTE**
>
> Director uses the file that you include in the **net_config_override** parameter as the template to generate the **/etc/os-net-config/config.yaml** file. **os-net-config** manages the interfaces that you define in the template, so you must perform all undercloud network interface customization in this file.

4. Install the undercloud.

Verification

- After the undercloud installation completes successfully, verify that the **/etc/os-net-config/config.yaml** file contains the relevant configuration:

```
network_config:
- name: br-ctlplane
  type: ovs_bridge
  use_dhcp: false
  dns_servers: 192.168.122.1
  domain: lab.example.com
  ovs_extra:
  - "br-set-external-id br-ctlplane bridge-id br-ctlplane"
  addresses:
  - ip_netmask: 172.20.0.1/26
  members:
  - type: interface
    name: nic2
```

4.8. 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:

   - **undercloud-passwords.conf** – A list of all passwords for the director services.

   - **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.

4.9. CONFIGURING THE CPU ARCHITECTURE FOR THE OVERCLOUD

Red Hat OpenStack Platform (RHOSP) configures the CPU architecture of an overcloud as x86_64 by default. You can also deploy overcloud Compute nodes on POWER (ppc64le) hardware. For the Compute node cluster, you can use the same architecture, or use a combination of x86_64 and ppc64le systems.

> **NOTE**
>
> The undercloud, Controller nodes, Ceph Storage nodes, and all other systems are supported only on x86_64 hardware.

4.9.1. Configuring POWER (ppc64le) as the single CPU architecture for the overcloud

The default CPU architecture of the Compute nodes on an overcloud is x86_64. To deploy overcloud Compute nodes on POWER (ppc64le) hardware, you can change the architecture to ppc64le.

> **NOTE**
>
> When your architecture includes POWER (ppc64le) nodes, RHOSP 16.1 supports only PXE boot.

Procedure

1. Disable iPXE in the **undercloud.conf** file:

   ```
   [DEFAULT]
   ipxe_enabled = False
   ```

   > **NOTE**
   >
   > This configuration causes any x86_64 nodes in your deployment to also boot in PXE/legacy mode.

2. Install the undercloud:

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

   For more information, see Installing director on the undercloud .

3. Wait until the installation script completes.

4. Obtain and upload the images for the overcloud nodes. For more information, see Obtaining images for overcloud nodes.

4.9.2. Using Ceph Storage in a multi-architecture overcloud

When you configure access to external Ceph in a multi-architecture cloud, set the
CephAnsiblePlaybook parameter to **/usr/share/ceph-ansible/site.yml.sample** and include your client
key and other Ceph-specific parameters.

For example:

```
parameter_defaults:
  CephAnsiblePlaybook: /usr/share/ceph-ansible/site.yml.sample
  CephClientKey: AQDLOh1VgEp6FRAAFzT7Zw+Y9V6JJExQAsRnRQ==
  CephClusterFSID: 4b5c8c0a-ff60-454b-a1b4-9747aa737d19
  CephExternalMonHost: 172.16.1.7, 172.16.1.8
```

4.9.3. Using composable services in a multi-architecture overcloud

The following services typically form part of the Controller node and are available for use in custom roles
as Technology Preview:

- Block Storage service (cinder)

- Image service (glance)

- Identity service (keystone)

- Networking service (neutron)

- Object Storage service (swift)

> **NOTE**
>
> Red Hat does not support features in Technology Preview.

For more information about composable services, see composable services and custom roles in the
Advanced Overcloud Customization guide. Use the following example to understand how to move the
listed services from the Controller node to a dedicated ppc64le node:

```
(undercloud) [stack@director ~]$ rsync -a /usr/share/openstack-tripleo-heat-templates/. ~/templates
(undercloud) [stack@director ~]$ cd ~/templates/roles
(undercloud) [stack@director roles]$ cat <<EO_TEMPLATE >ControllerPPC64LE.yaml
###############################################################################
# Role: ControllerPPC64LE                                      #
###############################################################################
- name: ControllerPPC64LE
  description: |
    Controller role that has all the controller services loaded and handles
    Database, Messaging and Network functions.
  CountDefault: 1
  tags:
    - primary
    - controller
  networks:
    - External
    - InternalApi
    - Storage
    - StorageMgmt
    - Tenant
```

```
# For systems with both IPv4 and IPv6, you may specify a gateway network for
# each, such as ['ControlPlane', 'External']
default_route_networks: ['External']
HostnameFormatDefault: '%stackname%-controllerppc64le-%index%'
ImageDefault: ppc64le-overcloud-full
ServicesDefault:
  - OS::TripleO::Services::Aide
  - OS::TripleO::Services::AuditD
  - OS::TripleO::Services::CACerts
  - OS::TripleO::Services::CephClient
  - OS::TripleO::Services::CephExternal
  - OS::TripleO::Services::CertmongerUser
  - OS::TripleO::Services::CinderApi
  - OS::TripleO::Services::CinderBackendDellPs
  - OS::TripleO::Services::CinderBackendDellSc
  - OS::TripleO::Services::CinderBackendDellEMCUnity
  - OS::TripleO::Services::CinderBackendDellEMCVMAXISCSI
  - OS::TripleO::Services::CinderBackendDellEMCVNX
  - OS::TripleO::Services::CinderBackendDellEMCXTREMIOISCSI
  - OS::TripleO::Services::CinderBackendNetApp
  - OS::TripleO::Services::CinderBackendScaleIO
  - OS::TripleO::Services::CinderBackendVRTSHyperScale
  - OS::TripleO::Services::CinderBackup
  - OS::TripleO::Services::CinderHPELeftHandISCSI
  - OS::TripleO::Services::CinderScheduler
  - OS::TripleO::Services::CinderVolume
  - OS::TripleO::Services::Collectd
  - OS::TripleO::Services::Docker
  - OS::TripleO::Services::Fluentd
  - OS::TripleO::Services::GlanceApi
  - OS::TripleO::Services::GlanceRegistry
  - OS::TripleO::Services::Ipsec
  - OS::TripleO::Services::Iscsid
  - OS::TripleO::Services::Kernel
  - OS::TripleO::Services::Keystone
  - OS::TripleO::Services::LoginDefs
  - OS::TripleO::Services::MySQLClient
  - OS::TripleO::Services::NeutronApi
  - OS::TripleO::Services::NeutronBgpVpnApi
  - OS::TripleO::Services::NeutronSfcApi
  - OS::TripleO::Services::NeutronCorePlugin
  - OS::TripleO::Services::NeutronDhcpAgent
  - OS::TripleO::Services::NeutronL2gwAgent
  - OS::TripleO::Services::NeutronL2gwApi
  - OS::TripleO::Services::NeutronL3Agent
  - OS::TripleO::Services::NeutronLbaasv2Agent
  - OS::TripleO::Services::NeutronLbaasv2Api
  - OS::TripleO::Services::NeutronLinuxbridgeAgent
  - OS::TripleO::Services::NeutronMetadataAgent
  - OS::TripleO::Services::NeutronML2FujitsuCfab
  - OS::TripleO::Services::NeutronML2FujitsuFossw
  - OS::TripleO::Services::NeutronOvsAgent
  - OS::TripleO::Services::NeutronVppAgent
  - OS::TripleO::Services::Ntp
  - OS::TripleO::Services::ContainersLogrotateCrond
  - OS::TripleO::Services::OpenDaylightOvs
```

```
    - OS::TripleO::Services::Rhsm
    - OS::TripleO::Services::RsyslogSidecar
    - OS::TripleO::Services::Securetty
    - OS::TripleO::Services::SensuClient
    - OS::TripleO::Services::SkydiveAgent
    - OS::TripleO::Services::Snmp
    - OS::TripleO::Services::Sshd
    - OS::TripleO::Services::SwiftProxy
    - OS::TripleO::Services::SwiftDispersion
    - OS::TripleO::Services::SwiftRingBuilder
    - OS::TripleO::Services::SwiftStorage
    - OS::TripleO::Services::Timezone
    - OS::TripleO::Services::TripleoFirewall
    - OS::TripleO::Services::TripleoPackages
    - OS::TripleO::Services::Tuned
    - OS::TripleO::Services::Vpp
    - OS::TripleO::Services::OVNController
    - OS::TripleO::Services::OVNMetadataAgent
    - OS::TripleO::Services::Ptp
EO_TEMPLATE
(undercloud) [stack@director roles]$ sed -i~ -e '/OS::TripleO::Services::\
(Cinder\|Glance\|Swift\|Keystone\|Neutron\)/d' Controller.yaml
(undercloud) [stack@director roles]$ cd ../
(undercloud) [stack@director templates]$ openstack overcloud roles generate \
    --roles-path roles -o roles_data.yaml \
    Controller Compute ComputePPC64LE ControllerPPC64LE BlockStorage ObjectStorage
CephStorage
```

4.10. 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 based on your CPU architecture. 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.

4.10.1. Single CPU architecture overcloud images

Your Red Hat OpenStack Platform (RHOSP) installation includes packages that provide you with the following overcloud images for director:

- **overcloud-full**
- **overcloud-full-initrd**
- **overcloud-full-vmlinuz**

These images are necessary for deployment of the overcloud with the default CPU architecture, x86–64. Importing these images 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** and **rhosp-director-images-ipa-x86_64** packages:

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

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/overcloud-full-latest-16.1.tar /usr/share/rhosp-director-images/ironic-python-agent-latest-16.1.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/

7. Verify that the images are uploaded:

   > (undercloud) [stack@director images]$ openstack image list
   > +--------------------------------------+------------------------+
   > | ID | Name |
   > +--------------------------------------+------------------------+
   > | ef793cd0-e65c-456a-a675-63cd57610bd5 | overcloud-full |
   > | 9a51a6cb-4670-40de-b64b-b70f4dd44152 | overcloud-full-initrd |
   > | 4f7e33f4-d617-47c1-b36f-cbe90f132e5d | overcloud-full-vmlinuz |
   > +--------------------------------------+------------------------+

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

4.10.2. Multiple CPU architecture overcloud images

Your Red Hat OpenStack Platform (RHOSP) installation includes packages that provide you with the following images that are necessary for deployment of the overcloud with the default CPU architecture, x86-64:

- **overcloud-full**

- **overcloud-full-initrd**

- **overcloud-full-vmlinuz**

Your RHOSP installation also includes packages that provide you with the following images that are necessary for deployment of the overcloud with the POWER (ppc64le) CPU architecture:

- **ppc64le-overcloud-full**

Importing these images 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-all** package:

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

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

   ```
   (undercloud) [stack@director ~]$ cd ~/images
   (undercloud) [stack@director images]$ for arch in x86_64 ppc64le ; do mkdir $arch ; done
   (undercloud) [stack@director images]$ for arch in x86_64 ppc64le ; do for i in
   /usr/share/rhosp-director-images/overcloud-full-latest-16.1-${arch}.tar /usr/share/rhosp-
   director-images/ironic-python-agent-latest-16.1-${arch}.tar ; do tar -C $arch -xf $i ; done ;
   done
   ```

5. Import the images into director:

   ```
   (undercloud) [stack@director ~]$ cd ~/images
   (undercloud) [stack@director images]$ openstack overcloud image upload --image-path
   ~/images/ppc64le --architecture ppc64le --whole-disk --http-boot
   /var/lib/ironic/tftpboot/ppc64le
   (undercloud) [stack@director images]$ openstack overcloud image upload --image-path
   ~/images/ppc64le --architecture ppc64le --whole-disk --image-type ironic-python-agent --
   http-boot /var/lib/ironic/httpboot/ppc64le
   (undercloud) [stack@director images]$ openstack overcloud image upload --image-path
   ~/images/x86_64/ --architecture x86_64 --http-boot /var/lib/ironic/tftpboot
   (undercloud) [stack@director images]$ openstack overcloud image upload --image-path
   ~/images/x86_64 --architecture x86_64 --image-type ironic-python-agent --http-boot
   /var/lib/ironic/httpboot
   ```

6. Verify that the images are uploaded:

```
(undercloud) [stack@director images]$ openstack image list
+--------------------------------------+-------------------------+--------+
| ID                                   | Name                    | Status |
+--------------------------------------+-------------------------+--------+
| 6a6096ba-8f79-4343-b77c-4349f7b94960 | overcloud-full          | active |
| de2a1bde-9351-40d2-bbd7-7ce9d6eb50d8 | overcloud-full-initrd   | active |
| 67073533-dd2a-4a95-8e8b-0f108f031092 | overcloud-full-vmlinuz  | active |
| f0fedcd0-3f28-4b44-9c88-619419007a03 | ppc64le-overcloud-full  | active |
+--------------------------------------+-------------------------+--------+
```

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

```
(undercloud) [stack@director images]$ ls -l /var/lib/ironic/tftpboot
/var/lib/ironic/tftpboot/ppc64le/
/var/lib/ironic/tftpboot:
total 422624
-rwxr-xr-x. 1 root  root     6385968 Aug  8 19:35 agent.kernel
-rw-r--r--. 1 root  root   425530268 Aug  8 19:35 agent.ramdisk
-rwxr--r--. 1 42422 42422      20832 Aug  8 02:08 chain.c32
-rwxr--r--. 1 42422 42422     715584 Aug  8 02:06 ipxe.efi
-rw-r--r--. 1 root  root         22 Aug  8 02:06 map-file
drwxr-xr-x. 2 42422 42422        62 Aug  8 19:34 ppc64le
-rwxr--r--. 1 42422 42422      26826 Aug  8 02:08 pxelinux.0
drwxr-xr-x. 2 42422 42422        21 Aug  8 02:06 pxelinux.cfg
-rwxr--r--. 1 42422 42422      69631 Aug  8 02:06 undionly.kpxe

/var/lib/ironic/tftpboot/ppc64le/:
total 457204
-rwxr-xr-x. 1 root  root  19858896 Aug  8 19:34 agent.kernel
-rw-r--r--. 1 root  root 448311235 Aug  8 19:34 agent.ramdisk
-rw-r--r--. 1 42422 42422      336 Aug  8 02:06 default
```

4.10.3. 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**
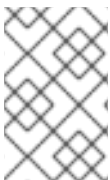
> **NOTE**
>
> The default **overcloud-full.qcow2** image is a flat partition image. However, you can also import and use whole disk images. For more information, see Chapter 24, *Creating whole-disk images*.

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-16.1.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
   ```

6. Verify that the images are uploaded:

   ```
   (undercloud) [stack@director images]$ openstack image list
   +--------------------------------------+--------------------------+
   | ID                                   | Name                     |
   +--------------------------------------+--------------------------+
   | ef793cd0-e65c-456a-a675-63cd57610bd5 | overcloud-full           |
   | 9a51a6cb-4670-40de-b64b-b70f4dd44152 | overcloud-full-initrd    |
   | 4f7e33f4-d617-47c1-b36f-cbe90f132e5d | overcloud-full-vmlinuz   |
   | 32cf6771-b5df-4498-8f02-c3bd8bb93fdd | overcloud-minimal        |
   | 600035af-dbbb-4985-8b24-a4e9da149ae5 | overcloud-minimal-initrd |
   | d45b0071-8006-472b-bbcc-458899e0d801 | overcloud-minimal-vmlinuz|
   +--------------------------------------+--------------------------+
   ```

4.11. SETTING A NAMESERVER FOR THE CONTROL PLANE

If you intend for the overcloud to resolve external hostnames, such as **cdn.redhat.com**, set a
nameserver on the overcloud nodes. For a standard overcloud without network isolation, the
nameserver is defined using the undercloud control plane subnet. Complete the following procedure to
define nameservers for the environment.

Procedure

1. Source the **stackrc** file to enable the director command line tools:

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

2. Set the nameservers for the **ctlplane-subnet** subnet:

```
(undercloud) [stack@director images]$ openstack subnet set --dns-nameserver
[nameserver1-ip] --dns-nameserver [nameserver2-ip] ctlplane-subnet
```

Use the **--dns-nameserver** option for each nameserver.

3. View the subnet to verify the nameserver:

```
(undercloud) [stack@director images]$ openstack subnet show ctlplane-subnet
+-------------------+----------------------------------------------+
| Field             | Value                                        |
+-------------------+----------------------------------------------+
| ...               |                                              |
| dns_nameservers   | 8.8.8.8                                      |
| ...               |                                              |
+-------------------+----------------------------------------------+
```

> **IMPORTANT**
>
> If you aim to isolate service traffic onto separate networks, the overcloud nodes must use the **DnsServers** parameter in your network environment files. You must also set the control plane nameserver and the **DnsServers** parameter to the same DNS server.

4.12. 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.

4.13. UNDERCLOUD CONTAINER REGISTRY

Red Hat Enterprise Linux 8.2 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.

CHAPTER 5. INSTALLING UNDERCLOUD MINIONS

You can deploy additional undercloud minions to scale OpenStack Platform director services across multiple hosts, which helps the performance when you deploy large overclouds. This feature is optional.

> **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.

5.1. UNDERCLOUD MINION

An undercloud minion provides additional **heat-engine** and **ironic-conductor** services on a separate host. These additional services support the undercloud with orchestration and provisioning operations. The distribution of undercloud operations across multiple hosts provides more resources to run an overcloud deployment, which can result in potentially faster and larger deployments.

5.2. UNDERCLOUD MINION REQUIREMENTS

Service requirements

The scaled **heat-engine** and **ironic-conductor** services on an undercloud minion use a set of workers. Each worker performs operations specific to that service. Multiple workers provide simultaneous operations. The default number of workers on the minion is determined by halving the total CPU thread count of the minion host. In this instance, total thread count is the number of CPU cores multiplied by the hyper-threading value. For example, if your minion has a CPU with 16 threads, then the minion spawns 8 workers for each service by default. The minion also uses a set of minimum and maximum caps by default:

Service	Minimum	Maximum
heat-engine	4	24
ironic-conductor	2	12

An undercloud minion 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 processor provides 4 workers for each undercloud service.

- A minimum of 16 GB of RAM.

To use a larger number of workers, increase the vCPUs and memory count on the undercloud using a ratio of 2 GB of RAM for each CPU thread. For example, a machine with 48 threads must have 96 GB of RAM. This provides coverage for 24 **heat-engine** workers and 12 **ironic-conductor** workers.

Container image requirements

An undercloud minion does not host an internal container image registry. As a result, you must configure the minion to use one of the following methods to obtain container images:

- Pull the images directly from the Red Hat Container Image Registry (**registry.redhat.io**).

- Pull images that you host on a Red Hat Satellite Server.

For both methods, you must to set **push_destination: false** as a part of the **ContainerImagePrepare** heat parameter in your **containers-prepare-parameter.yaml** file.

5.3. PREPARING A MINION

Before you can install a minion, you must complete some basic configuration on the host machine:

- A non-root user to execute commands.

- A resolvable hostname

- A Red Hat subscription

- The command line tools for image preparation and minion installation

Procedure

1. Log in to the minion host as the **root** user.

2. Create the **stack** user:

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

3. Set a password for the **stack** user:

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

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

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

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

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

6. Check the base and full hostname of the minion:

   ```
   [stack@minion ~]$ hostname
   [stack@minion ~]$ 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@minion ~]$ sudo hostnamectl set-hostname minion.example.com
   [stack@minion ~]$ sudo hostnamectl set-hostname --transient minion.example.com
   ```

7. Edit the **/etc/hosts** file and include an entry for the system hostname. For example, if the system is named **minion.example.com** and uses the IP address **10.0.0.1**, add the following line to the **/etc/hosts** file:

```
10.0.0.1  minion.example.com manager
```

8. Register your system either with the Red Hat Content Delivery Network or 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@minion ~]$ sudo subscription-manager register
```

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

```
[stack@minion ~]$ 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)
                     Red Hat Virtualization
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
```

10. Locate the **Pool ID** value and attach the Red Hat OpenStack Platform 16.1 entitlement:

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

11. Disable all default repositories, and then enable the required Red Hat Enterprise Linux repositories:

```
[stack@minion ~]$ sudo subscription-manager repos --disable=*
[stack@minion ~]$ sudo subscription-manager repos --enable=rhel-8-for-x86_64-baseos-
eus-rpms --enable=rhel-8-for-x86_64-appstream-eus-rpms --enable=rhel-8-for-x86_64-
highavailability-eus-rpms --enable=ansible-2.9-for-rhel-8-x86_64-rpms --enable=openstack-
16.1-for-rhel-8-x86_64-rpms --enable=fast-datapath-for-rhel-8-x86_64-rpms
```

These repositories contain packages that the minion installation requires.

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

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

13. Install the command line tools for minion installation and configuration:

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

5.4. COPYING THE UNDERCLOUD CONFIGURATION FILES TO THE MINION

The minion requires some configuration files from the undercloud so that the minion installation can configure the minion services and register them with director:

- **tripleo-undercloud-outputs.yaml**

- **tripleo-undercloud-passwords.yaml**

Procedure

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

2. Copy the files from the undercloud to the minion:

```
$ scp ~/tripleo-undercloud-outputs.yaml ~/tripleo-undercloud-passwords.yaml
stack@<minion-host>:~/.
```

   - Replace **<minion-host>** with the hostname or IP address of the minion.

5.5. COPYING THE UNDERCLOUD CERTIFICATE AUTHORITY

If the undercloud uses SSL/TLS for endpoint encryption, the minion host must contain the certificate authority that signed the undercloud SSL/TLS certificates. Depending on your undercloud configuration, this certificate authority is one of the following:

- An external certificate authority whose certificate is preloaded on the minion host. No action is required.

- A director-generated self-signed certificate authority, which the director creates at **/etc/pki/ca-trust/source/anchors/cm-local-ca.pem**. Copy this file to the minion host and include the file as a part of the trusted certificate authorities for the minion host. This procedure uses this file as an example.

- A custom self-signed certificate authority, which you create with OpenSSL. Examples in this document refer to this file as **ca.crt.pem**. Copy this file to the minion host and include the file as a part of the trusted certificate authorities for the minion host.

Procedure

1. Log in to the minion host as the **root** user.

2. Copy the certificate authority file from the undercloud to the minion:

```
[root@minion ~]# scp \
    root@<undercloud-host>:/etc/pki/ca-trust/source/anchors/cm-local-ca.pem \
    /etc/pki/ca-trust/source/anchors/undercloud-ca.pem
```

   - Replace **<undercloud-host>** with the hostname or IP address of the undercloud.

3. Update the trusted certificate authorities for the minion host:

```
[root@minion ~]# update-ca-trust enable
[root@minion ~]# update-ca-trust extract
```

5.6. CONFIGURING THE MINION

The minion installation process requires certain settings in the **minion.conf** configuration file, which the minion reads from the home directory of the **stack** user. Complete the following steps to use the default template as a foundation for your configuration.

Procedure

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

2. Copy the default template to the home directory of the **stack** user:

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

3. Edit the **minion.conf** file. This file contains settings to configure your minion. If you omit or comment out a parameter, the minion installation uses the default value. Review the following recommended parameters:

   - **minion_hostname**, which you set to the hostname of the minion.

   - **minion_local_interface**, which you set to the interface that connects to the undercloud through the Provisioning Network.

   - **minion_local_ip**, which you set to a free IP address on the Provisioning Network.

   - **minion_nameservers**, which you set to the DNS nameservers so that the minion can resolve hostnames.

   - **enable_ironic_conductor**, which defines whether to enable the **ironic-conductor** service.

   - **enable_heat_engine**, which defines whether to enable the **heat-engine** service.

> **NOTE**
>
> The default **minion.conf** file enables only the **heat-engine** service on the minion. To enable the **ironic-conductor** service, set the **enable_ironic_conductor** parameter to **true**.

5.7. MINION CONFIGURATION PARAMETERS

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

Defaults

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

cleanup

Cleanup temporary files. Set this parmaeter to **False** to leave the temporary files used during deployment in place after the command is run. This is useful for debugging the generated files or if errors occur.

container_cli

The CLI tool for container management. Leave this parameter set to **podman**. Red Hat Enterprise Linux 8.2 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 minion is registered to Satellite.

container_registry_mirror

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

custom_env_files

Additional environment file that you want to add to the minion installation.

deployment_user

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

enable_heat_engine

Defines whether to install the heat engine on the minion. The default is **true**.

enable_ironic_conductor

Defines whether to install the ironic conductor service on the minion. The default value is **false**. Set this value to **true** to enable the ironic conductor service.

heat_container_image

URL for the heat container image that you want to use. Leave unset.

heat_native

Use native heat templates. Leave as **true**.

hieradata_override

Path to **hieradata** override file that configures Puppet hieradata on the director, providing custom configuration to services beyond the **minion.conf** parameters. If set, the minion installation copies this file to the **/etc/puppet/hieradata** directory and sets it as the first file in the hierarchy.

minion_debug

Set this value to **true** to enable the **DEBUG** log level for minion services.

minion_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.

minion_enable_validations

Enable validation services on the minion.

minion_hostname

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

minion_local_interface

The chosen interface for the Provisioning NIC on the undercloud. This is also the device that the minion 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: eth0: <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 eth0
       valid_lft 3462sec preferred_lft 3462sec
    inet6 fe80::5054:ff:fe75:2409/64 scope link
       valid_lft forever preferred_lft forever
3: eth1: <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 **eth0** and the Provisioning NIC uses **eth1**, which is currently not configured. In this case, set the **local_interface** to **eth1**. The configuration script attaches this interface to a custom bridge defined with the **inspection_interface** parameter.

minion_local_ip

The IP address defined for the Provisioning NIC on the undercloud. This is also the IP address that the minion 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 the default IP address conflicts with an existing IP address or subnet in your environment.

minion_local_mtu

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

minion_log_file

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

minion_nameservers

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

minion_ntp_servers

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

minion_password_file

The file that contains the passwords for the minion to connect to undercloud services. Leave this parameter set to the **tripleo-undercloud-passwords.yaml** file copied from the undercloud.

minion_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.

minion_timezone

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

minion_undercloud_output_file

The file that contains undercloud configuration information that the minion can use to connect to undercloud services. Leave this parameter set to the **tripleo-undercloud-outputs.yaml** file copied from the undercloud.

net_config_override

The path to a network configuration override template. If you set this parameter, the minion uses a JSON format template to configure the networking with **os-net-config** and ignores the network parameters set in **minion.conf**. See **/usr/share/python-tripleoclient/minion.conf.sample** for an example.

networks_file

Networks file to override for **heat**.

output_dir

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

roles_file

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

templates

Heat templates file to override.

5.8. INSTALLING THE MINION

Complete the following steps to install the minion.

Procedure

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

2. Run the following command to install the minion:

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

   This command launches the configuration script for the minion, installs additional packages, and configures minion services according to the configuration in the **minion.conf** file. This script takes several minutes to complete.

5.9. VERIFYING THE MINION INSTALLATION

Complete the following steps to confirm a successful minion installation.

Procedure

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

2. Source the **stackrc** file:

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

3. If you enabled the heat engine service on the minion, verify that the **heat-engine** service from the minion appears on the undercloud service list:

   ```
   [stack@director ~]$ $ openstack orchestration service list
   ```

   The command output displays a table with **heat-engine** workers for both the undercloud and any minions.

4. If you enabled the ironic conductor service on the minion, verify that the **ironic-conductor** service from the minion appears on the undercloud service list:

   ```
   [stack@director ~]$ $ openstack baremetal conductor list
   ```

   The command output displays a table with **ironic-conductor** services for both the undercloud and any minions.

CHAPTER 6. 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.

6.1. 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 6.1. Node Deployment Roles for Scenarios

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

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 *Advanced Overcloud Customization* guide.

Table 6.2. 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.

6.2. 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

It is recommended that you deploy a project network (tunneled with GRE or VXLAN) even if you intend to use a neutron VLAN mode with tunneling disabled at deployment time. This requires minor customization at deployment time and leaves the option available to use tunnel networks as utility networks or virtualization networks in the future. You still create Tenant networks using VLANs, but you can also create VXLAN tunnels for special-use networks without consuming tenant VLANs. It is possible to add VXLAN capability to a deployment with a Tenant VLAN, but it is not possible to add a Tenant 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.

> **NOTE**
>
> You can virtualize the overcloud control plane if you are using Red Hat Virtualization (RHV). For more information, see Creating virtualized control planes.

6.3. OVERCLOUD STORAGE

> **NOTE**
>
> Using LVM on a guest instance that uses a back end cinder-volume of any driver or back-end type results in issues with performance, volume visibility and availability, and data corruption. Use an LVM filter to mitigate visibility, availability, and data corruption issues. For more information, see section 2 Block Storage and Volumes in the *Storage Guide* and KCS article 3213311, "Using LVM on a cinder volume exposes the data to the compute host."

Director includes different storage options for the overcloud environment:

Ceph Storage nodes

Director creates a set of scalable storage nodes using Red Hat Ceph Storage. The overcloud uses these nodes for the following storage types:

- **Images** – The Image service (glance) manages images for virtual machines. Images are immutable. OpenStack treats images as binary blobs and downloads them accordingly. You can use the Image service (glance) to store images in a Ceph Block Device.

- **Volumes** – OpenStack manages volumes with the Block Storage service (cinder). The Block Storage service (cinder) volumes are block devices. OpenStack uses volumes to boot virtual machines, or to attach volumes to running virtual machines. You can use the Block Storage service to boot a virtual machine using a copy-on-write clone of an image.

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

- **Guest Disks** – Guest disks are guest operating system disks. By default, when you boot a virtual machine with the Compute service (nova), the virtual machine disk appears as a file on the filesystem of the hypervisor (usually under **/var/lib/nova/instances/<uuid>/**). Every virtual machine inside Ceph can be booted without using the Block Storage service (cinder). As a result, you can perform maintenance operations easily with the live-migration process. Additionally, if your hypervisor fails, it is also convenient to trigger **nova evacuate** and run the virtual machine elsewhere.

> **IMPORTANT**
>
> For information about supported image formats, see Image Service in the *Creating and Managing Images* guide.

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

Swift Storage nodes

Director creates an external object storage node. 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.

6.4. 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 8

- Using SELinux for Red Hat Enterprise Linux 8

6.5. 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 High Availability Deployment and Usage .

> **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 High Availability for Compute Instances guide.

6.6. 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.

Red Hat provides several configuration recommendations for both Telemetry and Object Storage. For more information, see Deployment Recommendations for Specific Red Hat OpenStack Platform Services.

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.

Virtualization support

Red Hat supports virtualized Controller nodes only on Red Hat Virtualization platforms. For more information, see Creating virtualized control planes .

6.7. 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.

- IBM POWER 8 processor.

Memory

A minimum of 6 GB of RAM for the host operating system, plus additional memory to accomodate 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.

6.8. CEPH STORAGE NODE REQUIREMENTS

Ceph Storage nodes are responsible for providing object storage in a Red Hat OpenStack Platform environment.

For information about how to select a processor, memory, network interface cards (NICs), and disk layout for Ceph Storage nodes, see Hardware selection recommendations for Red Hat Ceph Storage in the *Red Hat Ceph Storage Hardware Guide* . Each Ceph Storage node also requires a supported power management interface, such as Intelligent Platform Management Interface (IPMI) functionality on the motherboard of the server.

NOTE

Red Hat OpenStack Platform (RHOSP) director uses **ceph-ansible**, which does not support installing the OSD on the root disk of Ceph Storage nodes. This means that you need at least two disks for a supported Ceph Storage node.

Ceph Storage nodes and RHEL compatibility

- RHOSP 16.1 is supported on RHEL 8.2. However, hosts that are mapped to the Ceph Storage role update to the latest major RHEL release. Before upgrading to RHOSP 16.1 and later, review the Red Hat Knowledgebase article Red Hat Ceph Storage: Supported configurations .

Placement Groups (PGs)

- Ceph Storage uses placement groups (PGs) to facilitate dynamic and efficient object tracking at scale. In the case of OSD failure or cluster rebalancing, Ceph can move or replicate a placement group and its contents, which means a Ceph Storage cluster can rebalance and recover efficiently.

- The default placement group count that director creates is not always optimal, so it is important to calculate the correct placement group count according to your requirements. You can use the placement group calculator to calculate the correct count. To use the PG calculator, enter the predicted storage usage per service as a percentage, as well as other properties about your Ceph cluster, such as the number OSDs. The calculator returns the optimal number of PGs per pool. For more information, see Placement Groups (PGs) per Pool Calculator.

- Auto-scaling is an alternative way to manage placement groups. With the auto-scale feature, you set the expected Ceph Storage requirements per service as a percentage instead of a specific number of placement groups. Ceph automatically scales placement groups based on how the cluster is used. For more information, see Auto-scaling placement groups in the *Red Hat Ceph Storage Strategies Guide*.

Processor

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

Network Interface Cards

- A minimum of one 1 Gbps Network Interface Cards (NICs), although Red Hat recommends that you use at least two NICs in a production environment. Use additional NICs for bonded interfaces or to delegate tagged VLAN traffic. Use a 10 Gbps interface for storage nodes, especially if you want to create a Red Hat OpenStack Platform (RHOSP) environment that serves a high volume of traffic.

Power management

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

For more information about installing an overcloud with a Ceph Storage cluster, see the Deploying an Overcloud with Containerized Red Hat Ceph guide.

6.9. 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.

6.10. OVERCLOUD REPOSITORIES

Red Hat OpenStack Platform (RHOSP) 16.1 runs on Red Hat Enterprise Linux 8.2. As a result, you must lock the content from these repositories to the respective Red Hat Enterprise Linux version.

NOTE

If you synchronize repositories by using Red Hat Satellite, you can enable specific versions of the Red Hat Enterprise Linux repositories. However, the repository label remains the same despite the version you choose. For example, if you enable the 8.2 version of the BaseOS repository, the repository name includes the specific version that you enabled, but the repository label is still **rhel-8-for-x86_64-baseos-tus-rpms**.

> **WARNING**
>
> Any repositories outside the ones specified here are not supported. Unless recommended, do not enable any other products or repositories outside 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 8.2 for x86_64 – BaseOS (RPMs) Telecommunications Update Service (TUS)	**rhel-8-for-x86_64-baseos-tus-rpms**	Base operating system repository for x86_64 systems.
Red Hat Enterprise Linux 8.2 for x86_64 – AppStream (RPMs)	**rhel-8-for-x86_64-appstream-tus-rpms**	Contains Red Hat OpenStack Platform dependencies.
Red Hat Enterprise Linux 8.2 for x86_64 – High Availability (RPMs) Telecommunications Update Service (TUS)	**rhel-8-for-x86_64-highavailability-tus-rpms**	High availability tools for Red Hat Enterprise Linux.
Red Hat Ansible Engine 2.9 for RHEL 8 x86_64 (RPMs)	**ansible-2.9-for-rhel-8-x86_64-rpms**	Ansible Engine for Red Hat Enterprise Linux. Used to provide the latest version of Ansible.
Advanced Virtualization for RHEL 8 x86_64 (RPMs)	**advanced-virt-for-rhel-8-x86_64-eus-rpms**	Provides virtualization packages for OpenStack Platform.
Red Hat OpenStack Platform 16.1 for RHEL 8 (RPMs)	**openstack-16.1-for-rhel-8-x86_64-rpms**	Core Red Hat OpenStack Platform repository.
Red Hat Fast Datapath for RHEL 8 (RPMS)	**fast-datapath-for-rhel-8-x86_64-rpms**	Provides Open vSwitch (OVS) packages for OpenStack Platform.
Red Hat Ceph Storage Tools 4 for RHEL 8 x86_64 (RPMs)	**rhceph-4-tools-for-rhel-8-x86_64-rpms**	Tools for Red Hat Ceph Storage 4 for Red Hat Enterprise Linux 8.
Red Hat Satellite Tools for RHEL 8 Server RPMs x86_64	**satellite-tools-6.5-for-rhel-8-x86_64-rpms**	Tools for managing hosts with Red Hat Satellite 6.

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 8.2 for x86_64 – BaseOS (RPMs) Telecommunications Update Service (TUS)	**rhel-8-for-x86_64-baseos-tus-rpms**	Base operating system repository for x86_64 systems.
Red Hat Enterprise Linux 8.2 for x86_64 – AppStream (RPMs)	**rhel-8-for-x86_64-appstream-tus-rpms**	Contains Red Hat OpenStack Platform dependencies.
Red Hat Enterprise Linux 8.2 for x86_64 – High Availability (RPMs) Telecommunications Update Service (TUS)	**rhel-8-for-x86_64-highavailability-tus-rpms**	High availability tools for Red Hat Enterprise Linux.
Red Hat Ansible Engine 2.9 for RHEL 8 x86_64 (RPMs)	**ansible-2.9-for-rhel-8-x86_64-rpms**	Ansible Engine for Red Hat Enterprise Linux. Used to provide the latest version of Ansible.
Advanced Virtualization for RHEL 8 x86_64 (RPMs)	**advanced-virt-for-rhel-8-x86_64-eus-rpms**	Provides virtualization packages for OpenStack Platform.
Red Hat OpenStack Platform 16.1 for RHEL 8 (RPMs)	**openstack-16.1-for-rhel-8-x86_64-rpms**	Core Red Hat OpenStack Platform repository.
Red Hat Fast Datapath for RHEL 8 (RPMS)	**fast-datapath-for-rhel-8-x86_64-rpms**	Provides Open vSwitch (OVS) packages for OpenStack Platform.
Red Hat Ceph Storage Tools 4 for RHEL 8 x86_64 (RPMs)	**rhceph-4-tools-for-rhel-8-x86_64-rpms**	Tools for Red Hat Ceph Storage 4 for Red Hat Enterprise Linux 8.
Red Hat Satellite Tools for RHEL 8 Server RPMs x86_64	**satellite-tools-6.5-for-rhel-8-x86_64-rpms**	Tools for managing hosts with Red Hat Satellite 6.

Real Time Compute repositories

The following table lists repositories for Real Time Compute (RTC) functionality.

Name	Repository	Description of requirement

Name	Repository	Description of requirement
Red Hat Enterprise Linux 8 for x86_64 – Real Time (RPMs)	**rhel-8-for-x86_64-rt-rpms**	Repository for Real Time KVM (RT-KVM). Contains packages to enable the real time kernel. Enable this repository for all Compute nodes targeted for RT-KVM. NOTE: You need a separate subscription to a **Red Hat OpenStack Platform for Real Time** SKU to access this repository.
Red Hat Enterprise Linux 8 for x86_64 – Real Time for NFV (RPMs)	**rhel-8-for-x86_64-nfv-rpms**	Repository for Real Time KVM (RT-KVM) for NFV. Contains packages to enable the real time kernel. Enable this repository for all NFV Compute nodes targeted for RT-KVM. NOTE: You need a separate subscription to a **Red Hat OpenStack Platform for Real Time** SKU to access this repository.

Ceph Storage node repositories

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

Name	Repository	Description of requirement
Red Hat Enterprise Linux 8.2 for x86_64 – BaseOS (RPMs) Telecommunications Update Service (TUS)	**rhel-8-for-x86_64-baseos-tus-rpms**	Base operating system repository for x86_64 systems.
Red Hat Enterprise Linux 8.2 for x86_64 – AppStream (RPMs)	**rhel-8-for-x86_64-appstream-tus-rpms**	Contains Red Hat OpenStack Platform dependencies.
Red Hat Enterprise Linux 8.2 for x86_64 – High Availability (RPMs) Telecommunications Update Service (TUS)	**rhel-8-for-x86_64-highavailability-tus-rpms**	High availability tools for Red Hat Enterprise Linux. NOTE: If you used the **overcloud-full** image for your Ceph Storage role, you must enable this repository. Ceph Storage roles should use the **overcloud-minimal** image, which does not require this repository.

Name	Repository	Description of requirement
Red Hat Ansible Engine 2.9 for RHEL 8 x86_64 (RPMs)	**ansible-2.9-for-rhel-8-x86_64-rpms**	Ansible Engine for Red Hat Enterprise Linux. Used to provide the latest version of Ansible.
Red Hat OpenStack Platform 16.1 Director Deployment Tools for RHEL 8 x86_64 (RPMs)	**openstack-16.1-deployment-tools-for-rhel-8-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-16.1-for-rhel-8-x86_64-rpms** repository.
Red Hat OpenStack Platform 16.1 for RHEL 8 (RPMs)	**openstack-16.1-for-rhel-8-x86_64-rpms**	Packages to help director configure Ceph Storage nodes. This repository is included with combined OpenStack Platform and Ceph Storage subscriptions. If you use a standalone Ceph Storage subscription, use the **openstack-16.1-deployment-tools-for-rhel-8-x86_64-rpms** repository.
Red Hat Ceph Storage Tools 4 for RHEL 8 x86_64 (RPMs)	**rhceph-4-tools-for-rhel-8-x86_64-rpms**	Provides tools for nodes to communicate with the Ceph Storage cluster.
Red Hat Fast Datapath for RHEL 8 (RPMS)	**fast-datapath-for-rhel-8-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.

IBM POWER repositories

The following table lists repositories for RHOSP on POWER PC architecture. Use these repositories in place of equivalents in the Core repositories.

Name	Repository	Description of requirement
Red Hat Enterprise Linux for IBM Power, little endian – BaseOS (RPMs)	**rhel-8-for-ppc64le-baseos-rpms**	Base operating system repository for ppc64le systems.

Name	Repository	Description of requirement
Red Hat Enterprise Linux 8 for IBM Power, little endian – AppStream (RPMs)	**rhel-8-for-ppc64le-appstream-rpms**	Contains Red Hat OpenStack Platform dependencies.
Red Hat Enterprise Linux 8 for IBM Power, little endian – High Availability (RPMs)	**rhel-8-for-ppc64le-highavailability-rpms**	High availability tools for Red Hat Enterprise Linux. Used for Controller node high availability.
Red Hat Fast Datapath for RHEL 8 IBM Power, little endian (RPMS)	**fast-datapath-for-rhel-8-ppc64le-rpms**	Provides Open vSwitch (OVS) packages for OpenStack Platform.
Red Hat Ansible Engine 2.8 for RHEL 8 IBM Power, little endian (RPMs)	**ansible-2.8-for-rhel-8-ppc64le-rpms**	Ansible Engine for Red Hat Enterprise Linux. Used to provide the latest version of Ansible.
Red Hat OpenStack Platform 16.1 for RHEL 8 (RPMs)	**openstack-16.1-for-rhel-8-ppc64le-rpms**	Core Red Hat OpenStack Platform repository for ppc64le systems.

6.11. PROVISIONING METHODS

There are three main methods that you can use to provision the nodes for your Red Hat OpenStack Platform environment:

Provisioning with director

Red Hat OpenStack Platform director is the standard provisioning method. In this scenario, the **openstack overcloud deploy** command performs both the provisioning and the configuration of your deployment. For more information about the standard provisioning and deployment method, see Chapter 7, *Configuring a basic overcloud* .

Provisioning with the OpenStack Bare Metal (ironic) service

In this scenario, you can separate the provisioning and configuration stages of the standard director deployment into two distinct processes. This is useful if you want to mitigate some of the risk involved with the standard director deployment and identify points of failure more efficiently. For more information about this scenario, see Chapter 8, *Provisioning bare metal nodes before deploying the overcloud*.

> **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.

Provisioning with an external tool

In this scenario, director controls the overcloud configuration on nodes that you pre-provision with an

external tool. 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 QCOW2 **overcloud-full** image. This scenario does not use the OpenStack Compute (nova), OpenStack Bare Metal (ironic), or OpenStack Image (glance) services for managing nodes.

For more information about this scenario, see Chapter 9, *Configuring a basic overcloud with pre-provisioned nodes*.

> **IMPORTANT**
>
> You cannot combine pre-provisioned nodes with director-provisioned nodes.

CHAPTER 7. CONFIGURING A BASIC OVERCLOUD

An overcloud with a basic configuration contains no custom features. To configure a basic Red Hat OpenStack Platform (RHOSP) environment, you must perform the following tasks:

- Register the bare-metal nodes for your overcloud.

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

- Tag each bare metal node with a resource class that matches the node to its designated role.

TIP

You can add advanced configuration options to this basic overcloud and customize it to your specifications. For more information, see Advanced Overcloud Customization.

7.1. 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": [{
      "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"
       }
      }],
      "name": "node01",
      "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"
     },
     {
      "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"
   }
  }],
  "name": "node02",
  "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:
  - 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
    name: "node01"
    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"
  - 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
    name: "node02"
    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.

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 Chapter 30, *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.

pm_port (Optional)

The port to access the specific IPMI device.

address

(Optional) A list of MAC addresses for the network interfaces on the node. Use only the MAC address for the Provisioning NIC of each system.

physical_network

(Optional) The physical network that is connected to the Provisioning NIC.

local_link_connection

(Optional) 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.

> **IMPORTANT**
>
> When building a multi-architecture cloud, the **arch** key is mandatory to distinguish nodes using **x86_64** and **ppc64le** architectures.

2. After you create the template, run the following commands to verify the formatting and syntax:

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

> **IMPORTANT**
>
> You must also include the **--http-boot** **/var/lib/ironic/tftpboot/** option for multi-architecture nodes.

3. Save the file to the home directory of the **stack** user (**/home/stack/nodes.json**), then run the following commands to import the template to director:

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

   This command registers each node from the template into director. If you use UEFI boot mode, you must also set the boot mode on each node. If you introspect your nodes without setting UEFI boot mode, the nodes boot in legacy mode. For more information, see Setting the boot mode to UEFI boot mode.

4. 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. 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.

7.2.1. 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)$ openstack tripleo validator run --group pre-introspection
   ```

4. Review the results of the validation report.

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

   ```
   (undercloud)$ openstack tripleo validator show run --full <validation>
   ```

   - Replace **<validation>** 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:

```
(undercloud)$ 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**:

```
(undercloud)$ 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:

```
(undercloud)$ 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. 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. Set the boot option to **local** for each registered node by adding **boot_option':'local** to the capabilities of the node:

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

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

4. 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**.

5. 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.

6. 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.

7. 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.

8. 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.

9. Validate the configuration of the node:

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

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

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

- **True**: The interface has passed validation.

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

7.3. TAGGING NODES INTO PROFILES

After you register and inspect the hardware of each node, tag the nodes into specific profiles. These profile tags match your nodes to flavors, which assigns the flavors to deployment roles. The following example shows the relationships across roles, flavors, profiles, and nodes for Controller nodes:

Type	Description
Role	The **Controller** role defines how director configures Controller nodes.

Type	Description
Flavor	The **control** flavor defines the hardware profile for nodes to use as controllers. You assign this flavor to the **Controller** role so that director can decide which nodes to use.
Profile	The **control** profile is a tag you apply to the**control** flavor. This defines the nodes that belong to the flavor.
Node	You also apply the **control** profile tag to individual nodes, which groups them to the **control** flavor and, as a result, director configures them using the **Controller** role.

Default profile flavors **compute**, **control**, **swift-storage**, **ceph-storage**, and **block-storage** are created during undercloud installation and are usable without modification in most environments.

Procedure

1. To tag a node into a specific profile, add a **profile** option to the **properties/capabilities** parameter for each node. For example, to tag a specific node to use a specific profile, use the following commands:

   ```
   (undercloud) $ NODE=<NODE NAME OR ID>
   (undercloud) $ PROFILE=<PROFILE NAME>
   (undercloud) $ openstack baremetal node set --property
   capabilities="profile:$PROFILE,boot_option:local" $NODE
   ```

   - Set the **$NODE** variable to the name or UUID of the node.

   - Set the **$PROFILE** variable to the specific profile, such as **control** or **compute**.

   - The **profile** option in **properties/capabilities** includes the **$PROFILE** variable to tag the node with the corresponding profile, such as **profile:control** or **profile:compute**.

   - Set the **boot_option:local** option to define how each node boots.

   You can also retain existing **capabilities** values using an additional **openstack baremetal node show** command and **jq** filtering:

   ```
   (undercloud) $ openstack baremetal node set --property
   capabilities="profile:$PROFILE,boot_option:local,$(openstack baremetal node show $NODE
   -f json -c properties | jq -r .properties.capabilities | sed "s/boot_mode:[^,]*,//g")" $NODE
   ```

2. After you complete node tagging, check the assigned profiles or possible profiles:

   ```
   (undercloud) $ openstack overcloud profiles list
   ```

7.4. SETTING THE BOOT MODE TO UEFI MODE

The default boot mode is Legacy BIOS mode. You can configure the nodes in your RHOSP deployment to use UEFI boot mode instead of Legacy BIOS boot mode.

> **WARNING**
>
> Some hardware does not support Legacy BIOS boot mode. If you attempt to use Legacy BIOS boot mode on hardware that does not support Legacy BIOS boot mode your deployment might fail. To ensure that your hardware deploys successfully, use UEFI boot mode.

> **NOTE**
>
> If you enable UEFI boot mode, you must build your own whole-disk image that includes a partitioning layout and bootloader, along with the user image. For more information about creating whole-disk images, see Creating whole-disk images.

Procedure

1. Set the following parameters in your **undercloud.conf** file:

   ```
   ipxe_enabled = True
   ```

2. Save the **undercloud.conf** file and run the undercloud installation:

   ```
   $ openstack undercloud install
   ```

   Wait until the installation script completes.

3. Check the existing capabilities of each registered node:

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

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

4. Set the boot mode to **uefi** for each registered node by adding **boot_mode:uefi** to the existing capabilities of the node:

   ```
   $ openstack baremetal node set --property capabilities="boot_mode:uefi,<capability_1>,...,
   <capability_n>" <node>
   ```

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

   - Replace **<capability_1>**, and all capabilities up to **<capability_n>**, with each capability that you retrieved in step 3.
     For example, use the following command to set the boot mode to **uefi** with local boot:

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

5. Set the boot mode to **uefi** for each bare metal flavor:

```
$ openstack flavor set --property capabilities:boot_mode='uefi' <flavor>
```

7.5. ENABLING VIRTUAL MEDIA BOOT
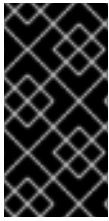
> **IMPORTANT**
>
> This feature is available in this release as a *Technology Preview*, and therefore is not fully supported by Red Hat. It should only be used for testing, and should not be deployed in a production environment. For more information about Technology Preview features, see Scope of Coverage 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 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, for UEFI nodes, define the EFI System Partition (ESP) image. Then configure an enrolled node to use Redfish virtual media boot.

Prerequisites

- Redfish driver enabled in the **enabled_hardware_types** parameter in the **undercloud.conf** file.

- A bare metal node registered and enrolled.

- IPA and instance images in the Image Service (glance).

- For UEFI nodes, you must also have an EFI system partition image (ESP) available in the Image Service (glance).

- A bare metal flavor.

- A network for cleaning and provisioning.

Procedure

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

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

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

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

```
NODE=<NODE NAME OR ID> ; openstack baremetal node set --property
capabilities="boot_mode:uefi,$(openstack baremetal node show $NODE -f json -c properties
| jq -r .properties.capabilities | sed "s/boot_mode:[^,]*,//g")" $NODE
```

- Replace **$NODE** with the name of the node.
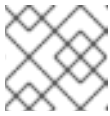
NOTE

For BIOS nodes, do not complete this step.

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

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

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

NOTE

For BIOS nodes, do not complete this step.

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

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

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

7.6. DEFINING THE ROOT DISK FOR MULTI-DISK CLUSTERS

Most Ceph Storage nodes use multiple disks. When nodes use multiple disks, director must identify the root disk. By default, director writes the overcloud image to the root disk during the provisioning process.

Use this procedure to identify the root device by serial number. For more information about other properties you can use to identify the root disk, see Section 7.7, "Properties that identify the root disk" .

Procedure

1. Verify the disk information from the hardware introspection of each node. The following command to displays the disk information of a node:

```
(undercloud)$ openstack baremetal introspection data save 1a4e30da-b6dc-499d-ba87-
0bd8a3819bc0 | jq ".inventory.disks"
```

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. On the undercloud, set the root disk for a node. Include the most appropriate hardware attribute value to define the root disk.

```
(undercloud)$ openstack baremetal node set --property root_device='{"serial":"
<serial_number>"}' <node-uuid>
```

For example, to set the root device to disk 2, which has the serial number **61866da04f380d001ea4e13c12e36ad6**, enter the following command:

```
(undercloud)$ openstack baremetal node set --property root_device='{"serial":
"61866da04f380d001ea4e13c12e36ad6"}' 1a4e30da-b6dc-499d-ba87-0bd8a3819bc0
```

> **NOTE**
>
> Configure the BIOS of each node to boot from the root disk that you choose.
> Configure the boot order to boot from the network first, then from the root disk.

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

7.7. 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 only for devices with persistent names. Do not use **name** to set the root disk for any other devices because this value can change when the node boots.

7.8. USING THE OVERCLOUD-MINIMAL IMAGE TO AVOID USING A RED HAT SUBSCRIPTION ENTITLEMENT

By default, director writes the QCOW2 **overcloud-full** image to the root disk during the provisioning process. The **overcloud-full** image uses a valid Red Hat subscription. However, you can also use the **overcloud-minimal** image, for example, to provision a bare OS where you do not want to run any other OpenStack services and consume your subscription entitlements.

A common use case for this occurs when you want to provision nodes with only Ceph daemons. For this and similar use cases, you can use the **overcloud-minimal** image option to avoid reaching the limit of your paid Red Hat subscriptions. For information about how to obtain the **overcloud-minimal** image, see Obtaining images for overcloud nodes .

> **NOTE**
>
> A Red Hat OpenStack Platform (RHOSP) subscription contains Open vSwitch (OVS), but core services, such as OVS, are not available when you use the **overcloud-minimal** image. 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 Linux bonding options.

Procedure

1. To configure director to use the **overcloud-minimal** image, create an environment file that contains the following image definition:

   ```
   parameter_defaults:
     <roleName>Image: overcloud-minimal
   ```

2. Replace **<roleName>** with the name of the role and append **Image** to the name of the role. The following example shows an **overcloud-minimal** image for Ceph storage nodes:

```
parameter_defaults:
  CephStorageImage: overcloud-minimal
```

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

```
rhsm_enforce: False
```

4. Pass the environment file to the **openstack overcloud deploy** command.

> **NOTE**
>
> The **overcloud-minimal** image supports only standard Linux bridges and not OVS because OVS is an OpenStack service that requires a Red Hat OpenStack Platform subscription entitlement.

7.9. CREATING ARCHITECTURE SPECIFIC ROLES

When building a multi-architecture cloud, you must add any architecture specific roles to the **roles_data.yaml** file. The following example includes the **ComputePPC64LE** role along with the default roles:

```
openstack overcloud roles generate \
    --roles-path /usr/share/openstack-tripleo-heat-templates/roles -o ~/templates/roles_data.yaml \
    Controller Compute ComputePPC64LE BlockStorage ObjectStorage CephStorage
```

The Creating a Custom Role File section has information on roles.

7.10. 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. However, the order of the environment files is important because the parameters and resources that you define in subsequent environment files take precedence. Use the following list as an example of the environment file order:

- The number of nodes and the flavors for each role. It is vital to include this information for overcloud creation.

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

- Any network isolation files, starting with the initialization file (**environments/network-isolation.yaml**) from the heat template collection, then your custom NIC configuration file, and finally any additional network configurations. For more information, see the following chapters in the *Advanced Overcloud Customization* guide:

    - "Basic network isolation"

    - "Custom composable networks"

    - "Custom network interface templates"

- Any external load balancing environment files if you are using an external load balancer. For more information, see External Load Balancing for the Overcloud .

- Any storage environment files such as Ceph Storage, NFS, or iSCSI.

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

- Any other custom environment files.

> **NOTE**
>
> Open Virtual Networking (OVN) is the default networking mechanism driver in Red Hat OpenStack Platform 16.1. 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.

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

For more information about customizing advanced features for your overcloud, see the Advanced Overcloud Customization guide.

> **IMPORTANT**
>
> A basic overcloud uses local LVM storage for block storage, which is not a supported configuration. It is recommended to use an external storage solution, such as Red Hat Ceph Storage, for block storage.

> **NOTE**
>
> The environment file extension must be **.yaml** or **.template**, or it will not be treated as a custom template resource.

The next few sections contain information about creating some environment files necessary for your overcloud.

7.11. CREATING AN ENVIRONMENT FILE THAT DEFINES NODE COUNTS AND FLAVORS

By default, director deploys an overcloud with 1 Controller node and 1 Compute node using the **baremetal** flavor. However, this is only suitable for a proof-of-concept deployment. You can override the default configuration by specifying different node counts and flavors. For a small-scale production environment, deploy at least 3 Controller nodes and 3 Compute nodes, and assign specific flavors to ensure that the nodes have the appropriate resource specifications. Complete the following steps to create an environment file named **node-info.yaml** that stores the node counts and flavor assignments.

Procedure

1. Create a **node-info.yaml** file in the **/home/stack/templates/** directory:

   ```
   (undercloud) $ touch /home/stack/templates/node-info.yaml
   ```

2. Edit the file to include the node counts and flavors that you need. This example contains 3 Controller nodes and 3 Compute nodes:

```
parameter_defaults:
  OvercloudControllerFlavor: control
  OvercloudComputeFlavor: compute
  ControllerCount: 3
  ComputeCount: 3
```

7.12. 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
BAYTAIVTMQswCQYDVQQIDAJOQzEQMA4GA1UEBwwHUmFsZWlnaDEQMA4GA1UECg
```

```
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.

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

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. For more information about environment files, see Section 7.16, "Including environment files in an overcloud deployment" .

7.13. 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)"

7.14. DEPLOYMENT COMMAND

The final stage in creating your OpenStack environment is to run the **openstack overcloud deploy** command to create the overcloud. Before you run this command, familiarize yourself with key options and how to include custom environment files.

> ⚠ **WARNING**
>
> Do not run **openstack overcloud deploy** as a background process. The overcloud creation might hang mid-deployment if you run it as a background process.

7.15. DEPLOYMENT COMMAND OPTIONS

The following table lists the additional parameters for the **openstack overcloud deploy** command.
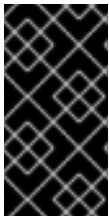
> **IMPORTANT**
>
> Some options are available in this release as a *Technology Preview* and therefore are not fully supported by Red Hat. They should only be used for testing and should not be used in a production environment. For more information about Technology Preview features, see Scope of Coverage Details.

Table 7.1. 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 **heat-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], **--extra-template [EXTRA HEAT TEMPLATE]**	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**.

Parameter	Description
-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**.
-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.

Parameter	Description
--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.
--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.

Parameter	Description
--limit NODE1,NODE2	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.
--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 7.2. Mapping deprecated CLI parameters to heat template parameters

Parameter	Description	Heat template parameter
--control-scale	The number of Controller nodes to scale out	**ControllerCount**
--compute-scale	The number of Compute nodes to scale out	**ComputeCount**
--ceph-storage-scale	The number of Ceph Storage nodes to scale out	**CephStorageCount**
--block-storage-scale	The number of Block Storage (cinder) nodes to scale out	**BlockStorageCount**
--swift-storage-scale	The number of Object Storage (swift) nodes to scale out	**ObjectStorageCount**
--control-flavor	The flavor that you want to use for Controller nodes	**OvercloudControllerFlavor**

Parameter	Description	Heat template parameter
--compute-flavor	The flavor that you want to use for Compute nodes	**OvercloudComputeFlavor**
--ceph-storage-flavor	The flavor that you want to use for Ceph Storage nodes	**OvercloudCephStorageFlavor**
--block-storage-flavor	The flavor that you want to use for Block Storage (cinder) nodes	**OvercloudBlockStorageFlavor**
--swift-storage-flavor	The flavor that you want to use for Object Storage (swift) nodes	**OvercloudSwiftStorageFlavor**
--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**
--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**

Parameter	Description	Heat template parameter
--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.

7.16. INCLUDING ENVIRONMENT FILES IN AN OVERCLOUD DEPLOYMENT

Use the **-e** option to include an environment file to customize your overcloud. You can include as many environment files as necessary. However, the order of the environment files is important because the parameters and resources that you define in subsequent environment files take precedence.

Any environment files that you add to the overcloud using the **-e** option become part of the stack definition of the overcloud.

The following command is an example of how to start the overcloud creation using environment files defined earlier in this scenario:

```
(undercloud) $ openstack overcloud deploy --templates \
  -e /home/stack/templates/node-info.yaml\
  -e /home/stack/containers-prepare-parameter.yaml \
```

```
-e /home/stack/inject-trust-anchor-hiera.yaml \
-r /home/stack/templates/roles_data.yaml \
```

This command contains the following additional options:

--templates

Creates the overcloud using the heat template collection in **/usr/share/openstack-tripleo-heat-templates** as a foundation.

-e /home/stack/templates/node-info.yaml

Adds an environment file to define how many nodes and which flavors to use for each role.

-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

(Optional) The generated roles data if you use custom roles or want to enable a multi architecture cloud. For more information, see Section 7.9, "Creating architecture specific roles" .

Director requires these environment files for re-deployment and post-deployment functions. Failure to include these files can result in damage to your overcloud.

To modify the overcloud configuration at a later stage, 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.

7.17. RUNNING THE PRE-DEPLOYMENT VALIDATION

Run the **pre-deployment** validation group to check the deployment requirements.

Procedure

1. Source the **stackrc** file.

   ```
   $ source ~/stackrc
   ```

2. This validation requires a copy of your overcloud plan. Upload your overcloud plan with all necessary environment files. To upload your plan only, run the **openstack overcloud deploy** command with the **--update-plan-only** option:

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

3. Run the **openstack tripleo validator run** command with the **--group pre-deployment** option:

```
$ openstack tripleo validator run --group pre-deployment
```

4. If the overcloud uses a plan name that is different to the default **overcloud** name, set the plan name with the --**plan** option:

```
$ openstack tripleo validator run --group pre-deployment \
    --plan myovercloud
```

5. Review the results of the validation report. To view detailed output from a specific validation, run the **openstack tripleo validator show run --full** command against the UUID of the specific validation from the report:

```
$ openstack tripleo validator show run --full <UUID>
```
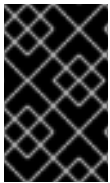
> **IMPORTANT**
>
> A **FAILED** validation does not prevent you from deploying or running Red Hat OpenStack Platform. However, a **FAILED** validation can indicate a potential issue with a production environment.

7.18. OVERCLOUD DEPLOYMENT OUTPUT

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 ******
========================================================================
```

Director also 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
```

7.19. ACCESSING THE OVERCLOUD

Director generates a script to configure and help authenticate interactions with your overcloud from the undercloud. Director saves this file, **overcloudrc**, in the home directory of the **stack** user. Run the following command to use this file:

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

This command loads the environment variables that are necessary to interact with your overcloud from the undercloud CLI. The command prompt changes to indicate this:

```
(overcloud) $
```

To return to interacting with the undercloud, run the following command:

```
(overcloud) $ source ~/stackrc
(undercloud) $
```

7.20. RUNNING THE POST-DEPLOYMENT VALIDATION

Run the **post-deployment** validation group to check the post-deployment state.

Procedure

1. Source the **stackrc** file.

   ```
   $ source ~/stackrc
   ```

2. Run the **openstack tripleo validator run** command with the **--group post-deployment** option:

   ```
   $ openstack tripleo validator run --group post-deployment
   ```

3. If the overcloud uses a plan name that is different to the default **overcloud** name, set the plan name with the --**plan** option:

   ```
   $ openstack tripleo validator run --group post-deployment \
       --plan myovercloud
   ```

4. Review the results of the validation report. To view detailed output from a specific validation, run the **openstack tripleo validator show run --full** command against the UUID of the specific validation from the report:

   ```
   $ openstack tripleo validator show run --full <UUID>
   ```
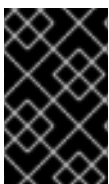
> **IMPORTANT**
>
> A **FAILED** validation does not prevent you from deploying or running Red Hat OpenStack Platform. However, a **FAILED** validation can indicate a potential issue with a production environment.

CHAPTER 8. PROVISIONING BARE METAL NODES BEFORE DEPLOYING THE OVERCLOUD

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

The overcloud deployment process contains two primary operations:

- Provisioning nodes

- Deploying the overcloud

You can mitigate some of the risk involved with this process and identify points of failure more efficiently if you separate these operations into distinct processes:

1. Provision your bare metal nodes.

   a. Create a node definition file in yaml format.

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

2. Deploy your overcloud.

   a. Run the deployment command, including the heat environment file that the provisioning command generates.

The provisioning process provisions your nodes and generates a heat environment file that contains various node specifications, including node count, predictive node placement, custom images, and custom NICs. When you deploy your overcloud, include this file in the deployment command.

> **IMPORTANT**
>
> You cannot combine pre-provisioned nodes with director-provisioned nodes.

8.1. 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": [{
       "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"
     }
    }],
    "name": "node01",
    "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"
   },
   {
    "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"
     }
    }],
    "name": "node02",
    "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:
 - 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
   name: "node01"
   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"
  - 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
  name: "node02"
  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.

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
> Chapter 30, *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.

pm_port (Optional)

The port to access the specific IPMI device.

address

(Optional) A list of MAC addresses for the network interfaces on the node. Use only the
MAC address for the Provisioning NIC of each system.

physical_network

(Optional) The physical network that is connected to the Provisioning NIC.

local_link_connection

(Optional) 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.

> **IMPORTANT**
>
> When building a multi-architecture cloud, the **arch** key is mandatory to distinguish nodes using **x86_64** and **ppc64le** architectures.

2. After you create the template, run the following commands to verify the formatting and syntax:

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

> **IMPORTANT**
>
> You must also include the **--http-boot /var/lib/ironic/tftpboot/** option for multi-architecture nodes.

3. Save the file to the home directory of the **stack** user (**/home/stack/nodes.json**), then run the following commands to import the template to director:

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

This command registers each node from the template into director. If you use UEFI boot mode, you must also set the boot mode on each node. If you introspect your nodes without setting UEFI boot mode, the nodes boot in legacy mode. For more information, see Setting the boot mode to UEFI boot mode.

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

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

8.2. 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.

8.2.1. 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)$ openstack tripleo validator run --group pre-introspection
   ```

4. Review the results of the validation report.

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

   ```
   (undercloud)$ openstack tripleo validator show run --full <validation>
   ```

   - Replace **<validation>** 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:

   ```
   (undercloud)$ 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**:

     ```
     (undercloud)$ 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:

```
(undercloud)$ 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.

8.2.2. 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. Set the boot option to **local** for each registered node by adding **boot_option':'local** to the capabilities of the node:

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

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

4. 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**.

5. 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.

6. 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.

7. 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.

8. 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.

9. Validate the configuration of the node:

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

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

   - **False**: The interface has failed validation. If the reason provided includes missing the **instance_info** parameters **['ramdisk', 'kernel', and 'image_source']**, this might be

because the Compute service populates those missing parameters at the beginning of the deployment process, therefore they have not been set at this point. If you are using a whole disk image, then you might need to only set **image_source** to pass the validation.

- **True**: The interface has passed validation.

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

8.3. PROVISIONING BARE METAL NODES

Create a new YAML file **~/overcloud-baremetal-deploy.yaml**, define the quantity and attributes of the bare metal nodes that you want to deploy, and assign overcloud roles to these nodes. The provisioning process creates a heat environment file that you can include in your **openstack overcloud deploy** command.

Prerequisites

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

- The bare metal nodes are 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 a new **~/overcloud-baremetal-deploy.yaml** 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, use the following syntax:

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

3. In the **~/overcloud-baremetal-deploy.yaml** file, define any predictive node placements, custom images, custom NICs, or other attributes that you want to assign to your nodes. For example, use the following example syntax 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
```

By default, the provisioning process uses the **overcloud-full** image. You can use the **image** attribute in the **instances** parameter to define a custom image:

```
- name: Controller
  count: 3
  instances:
  - hostname: overcloud-controller-0
    name: node00
    image:
      href: overcloud-custom
```

You can also override the default parameter values with the **defaults** parameter to avoid manual node definitions for each node entry:

```
- name: Controller
  count: 3
  defaults:
    image:
      href: overcloud-custom
  instances:
  - hostname :overcloud-controller-0
    name: node00
  - hostname: overcloud-controller-1
    name: node01
  - hostname: overcloud-controller-2
    name: node02
```

For more information about the parameters, attributes, and values that you can use in your node definition file, see Bare metal node provisioning attributes.

4. Run the provisioning command, specifying the ~/**overcloud-baremetal-deploy.yaml** file and defining an output file with the **--output** option:

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

The provisioning process generates a heat environment file with the name that you specify in the **--output** option. This file contains your node definitions. When you deploy the overcloud, include this file in the deployment command.

5. In a separate terminal, monitor your nodes to verify that they provision successfully. The provisioning process changes the node state from **available** to **active**:

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

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

```
(undercloud)$ metalsmith list
```

You can also use the **openstack baremetal allocation** command to verify association of nodes to hostnames:

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

When your nodes are provisioned successfully, you can deploy the overcloud. For more information, see Configuring a basic overcloud with pre-provisioned nodes .

8.4. 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

- A successful undercloud installation. For more information, see Installing director .

- A successful overcloud deployment. For more information, see Configuring a basic overcloud with pre-provisioned nodes.

- Bare metal nodes 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. Edit the ~/**overcloud-baremetal-deploy.yaml** file that you used to provision your bare metal nodes, and increment the **count** parameter for the roles that you want to scale up. For example, if your overcloud contains three Compute nodes, use the following snippet to increase the Compute node count to 10:

```
- name: Controller
  count: 3
- name: Compute
  count: 10
```

You can also add predictive node placement with the **instances** parameter. For more information about the parameters and attributes that are available, see Bare metal node provisioning attributes.

3. Run the provisioning command, specifying the ~/**overcloud-baremetal-deploy.yaml** file and defining an output file with the **--output** option:

```
(undercloud)$ openstack overcloud node provision \
--stack stack \
```

```
--output ~/overcloud-baremetal-deployed.yaml \
~/overcloud-baremetal-deploy.yaml
```

4. Monitor the provisioning progress with the **openstack baremetal node list** command.

5. Deploy the overcloud, including the ~/**overcloud-baremetal-deployed.yaml** file that the provisioning command generates, along with any other environment files relevant to your deployment:

```
(undercloud)$ openstack overcloud deploy \
  ...
  -e /usr/share/openstack-tripleo-heat-templates/environments/deployed-server-
environment.yaml \
  -e ~/overcloud-baremetal-deployed.yaml \
  --deployed-server \
  --disable-validations \
  ...
```

8.5. SCALING DOWN BARE METAL NODES

Tag the nodes that you want to delete from the stack in the ~/**overcloud-baremetal-deploy.yaml** file, redeploy the overcloud, and then include this file in the **openstack overcloud node delete** command with the **--baremetal-deployment** option.

Prerequisites

- A successful undercloud installation. For more information, see Chapter 4, *Installing director on the undercloud*.

- A successful overcloud deployment. For more information, see Chapter 9, *Configuring a basic overcloud with pre-provisioned nodes*.

- At least one bare metal node that you want to remove from the stack.

Procedure

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

   ```
   $ source ~/stackrc
   ```

2. Edit the ~/**overcloud-baremetal-deploy.yaml** file that you used to provision your bare metal nodes, and decrement the **count** parameter for the roles that you want to scale down. You must also define the following attributes for each node that you want to remove from the stack:

   - The name of the node.

   - The hostname that is associated with the node.

   - The attribute **provisioned: false**.
     For example, to remove the node **overcloud-controller-1** from the stack, include the following snippet in your ~/**overcloud-baremetal-deploy.yaml** file:

     ```
     - name: Controller
       count: 2
     ```

```
instances:
- hostname: overcloud-controller-0
  name: node00
- hostname: overcloud-controller-1
  name: node01
  # Removed from cluster due to disk failure
  provisioned: false
- hostname: overcloud-controller-2
  name: node02
```

3. Run the provisioning command, specifying the ~/**overcloud-baremetal-deploy.yaml** file and defining an output file with the **--output** option:

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

4. Redeploy the overcloud and include the ~/**overcloud-baremetal-deployed.yaml** file that the provisioning command generates, along with any other environment files relevant to your deployment:

```
(undercloud)$ openstack overcloud deploy \
  ...
  -e /usr/share/openstack-tripleo-heat-templates/environments/deployed-server-
environment.yaml \
  -e ~/overcloud-baremetal-deployed.yaml \
  --deployed-server \
  --disable-validations \
  ...
```

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**
>
> If you want to remove a node from the stack temporarily, you can 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.

5. Run the **openstack overcloud node delete** command, including the ~/**overcloud-baremetal-deploy.yaml** file with the **--baremetal-deployment** option.

```
(undercloud)$ openstack overcloud node delete \
--stack stack \
--baremetal-deployment ~/overcloud-baremetal-deploy.yaml
```

> **NOTE**
>
> Do not include the nodes that you want to remove from the stack as command arguments in the **openstack overcloud node delete** command.

8.6. BARE METAL NODE PROVISIONING ATTRIBUTES

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

Table 8.1. Role parameters

Parameter	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.
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 Table 8.2, "**instances** and **defaults** parameters". The length of this list must not be greater than the value of the **count** parameter.
hostname_format	Overrides the default hostname format for this role. The default format uses the lower case role name. 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%**

Example syntax

In the following example, the **name** refers to the logical name of the node, and the **hostname** refers to the generated hostname which is derived from the overcloud stack name, the role, and an incrementing index. All Controller servers use a default custom image **overcloud-full-custom** and are on predictive nodes. One of the Compute servers is placed predictively on **node04** with custom host name **overcloud-compute-special**, and the other 99 Compute servers are on nodes allocated automatically from the pool of available nodes:

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

```
    name: node02
 - name: Compute
   count: 100
   instances:
   - hostname: overcloud-compute-special
     name: node04
```

Table 8.2. **instances** and **defaults** parameters

Parameter	Value
hostname	If the hostname complies with the **hostname_format** pattern then other properties apply to the node allocated to this hostname. Otherwise, you can use a custom hostname for this node.
name	The name of the node that you want to provision.
image	Details of the image that you want to provision onto the node. For more information about supported properties in the **image** parameter, see Table 8.3, "**image** parameters".
capabilities	Selection criteria to match the node capabilities.
nics	List of dictionaries that represent requested NICs. For more information about supported properties in the **nics** parameter, see Table 8.4, "**nic** parameters".
profile	Selection criteria to use Advanced Profile Matching.
provisioned	Boolean to determine whether this node is provisioned or unprovisioned. The default value is **true**. Use **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**.
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.

Example syntax

In the following example, all Controller servers use a custom default overcloud image **overcloud-full-custom**. The Controller server **overcloud-controller-0** is placed predictively on **node00** and has custom

root and swap sizes. The other two Controller servers are on nodes allocated automatically from the pool of available nodes, and have default root and swap sizes:

```
- name: Controller
  count: 3
  defaults:
    image:
      href: file:///var/lib/ironic/images/overcloud-full-custom.qcow2
  instances:
  - hostname: overcloud-controller-0
    name: node00
    root_size_gb: 140
    swap_size_mb: 600
```

Table 8.3. **image** parameters

Parameter	Value
href	Glance image reference or URL of the root partition or whole disk image. URL schemes supported are **file://**, **http://**, and **https://**. If the value is not a valid URL, this value must be a valid glance image reference.
checksum	When the href is a URL, this value must be the SHA512 checksum of the root partition or whole disk image.
kernel	Glance image reference or URL of the kernel image. Use this property only for partition images.
ramdisk	Glance image reference or URL of the ramdisk image. Use this property only for partition images.

Example syntax
In the following example, all three Controller servers are on nodes allocated automatically from the pool of available nodes. All Controller servers in this environment use a default custom image **overcloud-full-custom**:

```
- name: Controller
  count: 3
  defaults:
    image:
      href: file:///var/lib/ironic/images/overcloud-full-custom.qcow2
      checksum: 1582054665
      kernel: file:///var/lib/ironic/images/overcloud-full-custom.vmlinuz
      ramdisk: file:///var/lib/ironic/images/overcloud-full-custom.initrd
```

Table 8.4. **nic** parameters

Parameter	Value
fixed_ip	The specific IP address that you want to use for this NIC.
network	The neutron network where you want to create the port for this NIC.
subnet	The neutron subnet where you want to create the port for this NIC.
port	Existing Neutron port to use instead of creating a new port.

Example syntax

In the following example, all three Controller servers are on nodes allocated automatically from the pool of available nodes. All Controller servers in this environment use a default custom image **overcloud-full-custom** and have specific networking requirements:

```
- name: Controller
  count: 3
  defaults:
    image:
      href: file:///var/lib/ironic/images/overcloud-full-custom.qcow2
    nics:
      network: custom-network
      subnet: custom-subnet
```

CHAPTER 9. 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. However, you can add advanced configuration options to this basic overcloud and customize it to your specifications with the instructions in the Advanced Overcloud Customization guide.

IMPORTANT

You cannot combine pre-provisioned nodes with director-provisioned nodes.

9.1. 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 8.2 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 9.1. 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 9.6, "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.

9.2. 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.

9.3. 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.

> **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:

```
[heat-admin@controller-0 ~]$ sudo subscription-manager register
```

2. Find the entitlement pool for Red Hat OpenStack Platform 16.1:

```
[heat-admin@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 Red Hat OpenStack Platform 16 entitlements:

```
[heat-admin@controller-0 ~]$ sudo subscription-manager attach --pool=pool_id
```

4. Disable all default repositories:

```
[heat-admin@controller-0 ~]$ sudo subscription-manager repos --disable=*
```

5. Enable the required Red Hat Enterprise Linux repositories.

   a. For x86_64 systems, run the following command:

   ```
   [heat-admin@controller-0 ~]$ sudo subscription-manager repos --enable=rhel-8-for-x86_64-baseos-eus-rpms --enable=rhel-8-for-x86_64-appstream-eus-rpms --enable=rhel-8-for-x86_64-highavailability-eus-rpms --enable=ansible-2.9-for-rhel-8-
   ```

> x86_64-rpms --enable=openstack-16.1-for-rhel-8-x86_64-rpms --enable=fast-datapath-for-rhel-8-x86_64-rpms --enable=advanced-virt-for-rhel-8-x86_64-rpms

b. For POWER systems, run the following command:

```
[heat-admin@controller-0 ~]$ sudo subscription-manager repos --enable=rhel-8-for-ppc64le-baseos-rpms --enable=rhel-8-for-ppc64le-appstream-rpms --enable=rhel-8-for-ppc64le-highavailability-rpms --enable=ansible-2.8-for-rhel-8-ppc64le-rpms --enable=openstack-16-for-rhel-8-ppc64le-rpms --enable=fast-datapath-for-rhel-8-ppc64le-rpms
```

6. Set the **container-tools** repository module to version **2.0**:

```
[heat-admin@controller-0 ~]$ sudo dnf module disable -y container-tools:rhel8
[heat-admin@controller-0 ~]$ sudo dnf module enable -y container-tools:2.0
```

7. If the overcloud uses Ceph Storage nodes, enable the relevant Ceph Storage repositories:

```
[heat-admin@cephstorage-0 ~]$ sudo subscription-manager repos --enable=rhel-8-for-x86_64-baseos-rpms --enable=rhel-8-for-x86_64-appstream-rpms --enable=rhel-8-for-x86_64-highavailability-rpms --enable=ansible-2.9-for-rhel-8-x86_64-rpms --enable=openstack-16.1-deployment-tools-for-rhel-8-x86_64-rpms
```

8. Lock each node to Red Hat Enterprise Linux 8.2 before you execute the **dnf update**:

```
[heat-admin@controller-0 ~]$ sudo subscription-manager release --set=8.2
```

9. Update your system to ensure you have the latest base system packages:

```
[heat-admin@controller-0 ~]$ sudo dnf update -y
[heat-admin@controller-0 ~]$ sudo reboot
```

The node is now ready to use for your overcloud.

9.4. 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.

9.5. 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 9.6, "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. There are multiple network isolation strategies in the the Advanced Overcloud Customization guide. You can also define specific IP addresses for nodes on the Control Plane. For more information about isolating networks and creating predictable node placement strategies, see the following sections in the *Advanced Overcloud Customizations* guide:

- "Basic network isolation"

- "Controlling Node Placement"

> **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.

For example, you can use an environment file **ctlplane-assignments.yaml** with the following IP assignments to implement a predictable IP strategy:

```
resource_registry:
  OS::TripleO::DeployedServer::ControlPlanePort: /usr/share/openstack-tripleo-heat-
templates/deployed-server/deployed-neutron-port.yaml
```

```
parameter_defaults:
  DeployedServerPortMap:
    controller-0-ctlplane:
      fixed_ips:
        - ip_address: 192.168.24.2
      subnets:
        - cidr: 192.168.24.0/24
      network:
        tags:
          192.168.24.0/24
    compute-0-ctlplane:
      fixed_ips:
        - ip_address: 192.168.24.3
      subnets:
        - cidr: 192.168.24.0/24
      network:
        tags:
          - 192.168.24.0/24
```

In this example, the **OS::TripleO::DeployedServer::ControlPlanePort** resource passes a set of parameters to director and defines the IP assignments of your pre-provisioned nodes. Use the **DeployedServerPortMap** parameter to define the IP addresses and subnet CIDRs that correspond to each overcloud node. The mapping defines the following attributes:

1. The name of the assignment, which follows the format **<node_hostname>-<network>** where the **<node_hostname>** value matches the short host name for the node, and **<network>** matches the lowercase name of the network. For example: **controller-0-ctlplane** for **controller-0.example.com** and **compute-0-ctlplane** for **compute-0.example.com**.

2. The IP assignments, which use the following parameter patterns:

   - **fixed_ips/ip_address** – Defines the fixed IP addresses for the control plane. Use multiple **ip_address** parameters in a list to define multiple IP addresses.

   - **subnets/cidr** – Defines the CIDR value for the subnet.

A later section in this chapter uses the resulting environment file (**ctlplane-assignments.yaml**) as part of the **openstack overcloud deploy** command.

9.6. 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 9.5, "Configuring networking for the control plane".

- You must enable SSL/TLS on the director for Public API endpoint usage. For more information, see Section 4.2, "Director configuration parameters" and Chapter 20, *Configuring custom SSL/TLS certificates*.

- 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 9.2. 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 9.5, "Configuring networking for the control plane". However, since the Control Plane is not routable from the deployed servers, you must use the **DeployedServerPortMap** parameter 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 **ctlplane-assignments.yaml** environment file from Section 9.5, "Configuring networking for the control plane" that accommodates this network architecture:

```
resource_registry:
  OS::TripleO::DeployedServer::ControlPlanePort: /usr/share/openstack-tripleo-heat-
templates/deployed-server/deployed-neutron-port.yaml
  OS::TripleO::Network::Ports::ControlPlaneVipPort: /usr/share/openstack-tripleo-heat-
templates/deployed-server/deployed-neutron-port.yaml
  OS::TripleO::Network::Ports::RedisVipPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/noop.yaml
  OS::TripleO::Network::Ports::OVNDBsVipPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/noop.yaml 1

parameter_defaults:
  NeutronPublicInterface: eth1
  DeployedServerPortMap:
    control_virtual_ip:
      fixed_ips:
        - ip_address: 192.168.100.1
      subnets:
        - cidr: 24
    controller-0-ctlplane:
      fixed_ips:
        - ip_address: 192.168.100.2
```

```
    subnets:
      - cidr: 24
  compute-0-ctlplane:
    fixed_ips:
      - ip_address: 192.168.100.3
    subnets:
      - cidr: 24
```

1 The **RedisVipPort** and **OVNDBsVipPort** resources are mapped to **network/ports/noop.yaml**. This mapping is necessary because the default Redis and OVNDBs VIP addresses come from the Control Plane. In this situation, use a **noop** to disable this Control Plane mapping.

9.7. 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.

9.8. MAPPING NETWORK INTERFACES TO ALIASES

In Red Hat OpenStack Platform 16.1, overcloud network interface mapping does not happen automatically on pre-provisioned nodes. Instead, you must define the mapping manually in the **/etc/os-net-config/mapping.yaml** file on each pre-provisioned node.

Procedure

1. Log in to each pre-provisioned node.

2. Create the **/etc/os-net-config/mapping.yaml** file and include the details of your interface mapping:

```
interface_mapping:
  nic1: em1
  nic2: em2
```

9.9. CONFIGURING CEPH STORAGE FOR PRE-PROVISIONED NODES

Complete the following steps on the undercloud host to configure **ceph-ansible** 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.

9.10. CREATING THE OVERCLOUD WITH PRE-PROVISIONED NODES

The overcloud deployment uses the standard CLI methods from Section 7.14, "Deployment command". 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/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 or the Workflow service (mistral), and only the director **openstack overcloud deploy** command uses this key to enable access for the **tripleo-admin** user.

9.11. OVERCLOUD DEPLOYMENT OUTPUT

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 ******
===========================================================================
```

Director also 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
```

9.12. ACCESSING THE OVERCLOUD

Director generates a script to configure and help authenticate interactions with your overcloud from the undercloud. Director saves this file, **overcloudrc**, in the home directory of the **stack** user. Run the following command to use this file:

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

This command loads the environment variables that are necessary to interact with your overcloud from the undercloud CLI. The command prompt changes to indicate this:

```
(overcloud) $
```

To return to interacting with the undercloud, run the following command:

```
(overcloud) $ source ~/stackrc
(undercloud) $
```

9.13. SCALING PRE-PROVISIONED NODES

The process for scaling pre-provisioned nodes is similar to the standard scaling procedures in Chapter 16, *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 OpenStack Bare Metal (ironic) and OpenStack Compute (nova).

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.

Perform the following actions to scale up overcloud nodes:

1. Prepare the new pre-provisioned nodes according to Section 9.1, "Pre-provisioned node requirements".

2. Scale up the nodes. For more information, see Chapter 16, *Scaling overcloud nodes*.

3. After you execute the deployment command, wait until the director creates the new node resources and launches the configuration.

Scaling down pre-provisioned nodes

When scaling down the overcloud with pre-provisioned nodes, follow the scale down instructions in Chapter 16, *Scaling overcloud nodes*.

In scale-down operations, you can use hostnames for both OSP provisioned or pre-provisioned nodes. You can also use the UUID for OSP provisioned nodes. However, there is no UUID for pre-provisoned nodes, so you always use hostnames. Pass the hostname or UUID value to the **openstack overcloud node delete** command.

Procedure

1. Identify the name of the node that you want to remove.

   ```
   $ openstack stack resource list overcloud -n5 --filter
   type=OS::TripleO::ComputeDeployedServerServer
   ```

2. Pass the corresponding node name from the **stack_name** column to the **openstack overcloud node delete** command:

   ```
   $ openstack overcloud node delete --stack <overcloud> <stack>
   ```

   - Replace **<overcloud>** with the name or UUID of the overcloud stack.

   - Replace **<stack_name>** with the name of the node that you want to remove. You can include multiple node names in the **openstack overcloud node delete** command.

3. Ensure that the **openstack overcloud node delete** command runs to completion:

   ```
   $ openstack stack list
   ```

The status of the **overcloud** stack shows **UPDATE_COMPLETE** when the delete operation is complete.

After you remove overcloud nodes from the stack, power off these nodes. In a standard deployment, the bare metal services on the director control this function. However, with pre-provisioned nodes, you must either manually shut down these 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.

After you power off the removed nodes, reprovision them 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.

Removing a pre-provisioned overcloud

To remove an entire overcloud that uses pre-provisioned nodes, see Section 12.6, "Removing the overcloud" for the standard overcloud remove 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 10. DEPLOYING MULTIPLE OVERCLOUDS
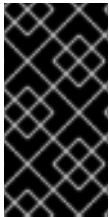
IMPORTANT

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 a single undercloud node to deploy and manage multiple overclouds. Each overcloud is a unique heat stack that does not share stack resources. This can be useful for environments where a 1:1 ratio of underclouds to overclouds creates an unmanageable amount of overhead. For example, Edge, multi-site, and multi-product environments.

The overcloud environments in a multiple overcloud deployment are completely separate, and you can use the **source** command to switch between the environments. Each overcloud has a unique credential file which is created by the deployment process. To interact with an overcloud, you must source the appropriate credential file.

If you use the Bare Metal Provisioning service (ironic) for bare metal provisioning, all overclouds must be on the same provisioning network. If it is not possible to use the same provisioning network, you can use the deployed servers method to deploy multiple overclouds with routed networks. In this scenario, you must ensure that the value in the **HostnameMap** parameter matches the stack name for each overcloud.

To deploy multiple overclouds on a single undercloud, you must perform the following tasks:

1. Deploy the undercloud. For more information, Part I. Director installation and configuration.

2. Deploy your first overcloud. For more information, see Part II. Basic overcloud deployment.

3. Deploy your additional overclouds by creating a new set of environment files for the new overcloud and specifying the core heat templates together with the new configuration files and a new **stack** name in the deployment command.

10.1. DEPLOYING AN ADDITIONAL OVERCLOUD

You can deploy more than one overcloud on a single undercloud. The following procedure illustrates how to create and deploy a new overcloud, **overcloud-two**, on an existing Red Hat OpenStack Platform (RHOSP) deployment that has an existing overcloud, **overcloud-one**.

Prerequisites

- An undercloud.

- One or more overclouds.

- Nodes that are available for your additional overcloud.

- Custom networks for additional overclouds so that each overcloud has a unique network in the resulting stack.

Procedure

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

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

   ```
   $ source ~/stackrc
   ```

3. Create a new directory for the additional overcloud that you want to deploy:

   ```
   (undercloud)$ mkdir ~/overcloud-two
   ```

4. Copy the **network_data.yaml** file from the existing overcloud to the new directory for the additional overcloud:

   ```
   (undercloud)$ cp network_data.yaml ~/overcloud-two/network_data.yaml
   ```

5. Open the ~/**overcloud-two/network_data.yaml** file and update **name_lower** to a unique name for the the additional overcloud network:

   ```
   - name: InternalApi
     name_lower: internal_api_cloud_2
     ...
   ```

6. Add **service_net_map_replace** if not already present, and set the value to the default value of the additional overcloud network:

   ```
   - name: InternalApi
     name_lower: internal_api_cloud_2
     service_net_map_replace: internal_api
   ```

7. Specify the VLAN IDs for each subnet on the additional overcloud:

   ```
   - name: InternalApi
     ...
     vip: true
     vlan: 21
     ip_subnet: '172.21.0.0/24'
     allocation_pools: [{'start': '172.21.0.4', 'end': '172.21.0.250'}]
     ipv6_subnet: 'fd00:fd00:fd00:2001::/64'
     ipv6_allocation_pools: [{'start': 'fd00:fd00:fd00:2001::10', 'end':
   'fd00:fd00:fd00:2001:ffff:ffff:ffff:fffe'}]
     mtu: 1500
   - name: Storage
     ...
   ```

8. Specify the IP address of the gateway for the **overcloud-two** external network:

   ```
   - name: External
     ...
     gateway_ip: <ip_address>
     ...
   ```

   - Replace **<ip_address>** with the IP address of the gateway for the **overcloud-two** external network, for example, **10.0.10.1**.

9. Create a network configuration file for the additional overcloud that overrides the default isolated network configuration provided in the **/usr/share/openstack-tripleo-heat-templates/environments/network-environment.yaml** file, for example, **network_overrides.yaml**.

10. Open the ~/**overcloud-two/network_overrides.yaml** file and add the IP address of the **overcloud-two** DNS server:

    ```
    parameter_defaults:
     ...
     DnsServers:
      - <ip_address>
     ...
    ```

    - Replace **<ip_address>** with the IP address of the **overcloud-two** DNS server, for example, **10.0.10.2**.

11. If your deployment uses predictable IP addresses, then configure the IP addresses for the **overcloud-two** nodes in a new network IP address mapping file, **ips-from-pool-overcloud-two.yaml**:

    ```
    parameter_defaults:
     ControllerIPs:
     ...
      internal_api_cloud_2:
      - 192.168.1.10
      - 192.168.1.11
      - 192.168.1.12

      ...
      external_cloud_2:
      - 10.0.1.41

      ...
    ```

12. Add your **overcloud-two** environment files to the stack with your other environment files and deploy the additional overcloud:

    ```
    (undercloud)$ openstack overcloud deploy --templates \
      --stack overcloud-two \
      -n ~/overcloud-two/network_data.yaml \
      -e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml \
      -e /usr/share/openstack-tripleo-heat-templates/environments/network-environment.yaml \
      -e /usr/share/openstack-tripleo-heat-templates/environments/net-single-nic-with-vlans.yaml \
      -e ~/overcloud-two/network_overrides.yaml \
      -e [your environment files] \
      ...
    ```

    The deployment process creates **overcloud-tworc** for interacting with and managing **overcloud-two**.

13. To interact with your additional overcloud, source the overcloud credential file:

    ```
    $ source overcloud-tworc
    ```

10.2. MANAGING MULTIPLE OVERCLOUDS

Each overcloud that you deploy uses the same set of core heat templates **/usr/share/openstack-tripleo-heat-templates**. Red Hat recommends that you do not modify or duplicate these templates, because using a non-standard set of core templates can introduce issues with updates and upgrades.

Instead, for ease of management when you deploy or maintain multiple overclouds, create separate directories of environment files specific to each cloud. When you run the deploy command for each cloud, include the core heat templates together with the cloud-specific environment files that you create separately. For example, create the following directories for the undercloud and two overclouds:

~stack/undercloud

Contains the environment files specific to the undercloud.

~stack/overcloud-one

Contains the environment files specific to the first overcloud.

~stack/overcloud-two

Contains the environment files specific to the second overcloud.

When you deploy or redeploy **overcloud-one** or **overcloud-two**, include the core heat templates in the deploy command with the **--templates** option, and then specify any additional environment files from the cloud-specific environment file directories.

Alternatively, create a repository in a version control system and use branches for each deployment. For more information, see the Using Customized Core Heat Templates section of the *Advanced Overcloud Customization guide*.

Use the following command to view a list of overcloud plans that are available:

```
$ openstack overcloud plan list
```

Use the following command to view a list of overclouds that are currently deployed:

```
$ openstack stack list
```

CHAPTER 11. 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.

11.1. 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 **--plan** argument to select an overcloud with a different name:

   ```
   $ openstack overcloud status --plan my-deployment
   ```

11.2. 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.

11.3. 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.

11.4. 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 11.7, "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.

11.5. 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**.

   These example commands create a shared network. It is also possible to specify a tenant instead of specifying **--share** so that only the tenant has access to the new network.

   + If you mark a provider network as external, only the operator may create ports on that 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.

11.6. CREATING ADDITIONAL BRIDGE MAPPINGS

Floating IP networks can use any bridge, not just **br-ex**, provided that you map the additional bridge during deployment.

For example, 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"
```

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
```

11.7. 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 OpenStack Integration Test Suite Guide.

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 OpenStack Integration Test Suite Guide .

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
```

11.8. PROTECTING THE OVERCLOUD FROM REMOVAL

Set a custom policy for heat to protect your overcloud from being deleted.

Procedure

1. Create an environment file called **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'
```

> **IMPORTANT**
>
> The **heat-deny-action** is a default policy that you must include in your undercloud installation.

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
```

This environment file prevents you from deleting any stacks in the overcloud, which means you cannot perform the following functions:

- Delete the overcloud

- Remove individual Compute nor Ceph Storage nodes

- Replace Controller nodes

To enable stack deletion, remove the **prevent-stack-delete.yaml** file from the **custom_env_files** parameter and run the **openstack undercloud install** command.

CHAPTER 12. PERFORMING BASIC OVERCLOUD ADMINISTRATION TASKS

This chapter contains information about basic tasks you might need to perform during the lifecycle of your overcloud.

12.1. ACCESSING OVERCLOUD NODES THROUGH SSH

You can access each overcloud node through the SSH protocol.

- Each overcloud node contains a **heat-admin** user.

- The **stack** user on the undercloud has key-based SSH access to the **heat-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. Source the **overcloudrc** file:

   ```
   $ source ~/stackrc
   ```

3. Find the name of the node that you want to access:

   ```
   (undercloud) $ openstack server list
   ```

4. Connect to the node as the **heat-admin** user and use the short hostname of the node:

   ```
   (undercloud) $ ssh heat-admin@overcloud-controller-0.ctlplane
   ```

12.2. 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. In OpenStack Platform 16, 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 20:18:33 UTC  4s left        Mon 2019-02-18 20:17:03 UTC  1min 25s ago
tripleo_mistral_engine_healthcheck.timer               tripleo_mistral_engine_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

OpenStack Platform 16 introduces a new logging directory **/var/log/containers/stdout** that contains the standard output (stdout) all of the containers, and standard errors (stderr) consolidated in one single file for each container.

Paunch and the **container-puppet.py** script configure podman containers to push their outputs to the **/var/log/containers/stdout** directory, which creates a collection of all logs, even for the deleted containers, such as **container-puppet-*** containers.

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
```

12.3. MODIFYING THE OVERCLOUD ENVIRONMENT

You can modify the overcloud to add additional features or alter existing operations. 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 7.14, "Deployment command", rerun the following command:

```
$ source ~/stackrc
(undercloud) $ openstack overcloud deploy --templates \
  -e ~/templates/node-info.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml \
  -e ~/templates/network-environment.yaml \
  -e ~/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.

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 /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml \
  -e ~/templates/network-environment.yaml \
  -e ~/templates/storage-environment.yaml \
  -e ~/templates/node-info.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.

12.4. 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 instance_name --name image_name
$ openstack image save image_name --file exported_vm.qcow2
```

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** file:

```
$ source ~/overcloudrc
```

5. Upload the exported image into the overcloud:

```
(overcloud) $ openstack image create imported_image --file exported_vm.qcow2 --disk-format qcow2 --container-format bare
```

6. Launch a new instance:

```
(overcloud) $ openstack server create  imported_instance --key-name default --flavor m1.demo --image imported_image --nic net-id=net_id
```

> **IMPORTANT**
>
> These commands copy each virtual machine disk from the existing OpenStack environment to the new Red Hat OpenStack Platform. QCOW snapshots lose their original layering system.

This process migrates all instances from a Compute node. You can now perform maintenance on the node without any instance downtime. To return the Compute node to an enabled state, run the following command:

```
$ source ~/overcloudrc
(overcloud) $ openstack compute service set [hostname] nova-compute --enable
```

12.5. RUNNING THE DYNAMIC INVENTORY SCRIPT

Director can run Ansible-based automation in your Red Hat OpenStack Platform (RHOSP) environment. Director uses the **tripleo-ansible-inventory** command to generate a dynamic inventory of nodes in your environment.

Procedure

1. To view a dynamic inventory of nodes, run the **tripleo-ansible-inventory** command after sourcing **stackrc**:

```
$ source ~/stackrc
(undercloud) $ tripleo-ansible-inventory --list
```

Use the **--list** option to return details about all hosts. This command outputs the dynamic inventory in a JSON format:

```
{"overcloud": {"children": ["controller", "compute"], "vars": {"ansible_ssh_user": "heat-admin"}},
"controller": ["192.168.24.2"], "undercloud": {"hosts": ["localhost"], "vars":
{"overcloud_horizon_url": "http://192.168.24.4:80/dashboard", "overcloud_admin_password":
"abcdefghijklm12345678", "ansible_connection": "local"}}, "compute": ["192.168.24.3"]}
```

2. To execute Ansible playbooks on your environment, run the **ansible** command and include the full path of the dynamic inventory tool using the **-i** option. For example:

```
(undercloud) $ ansible [HOSTS] -i /bin/tripleo-ansible-inventory [OTHER 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

  - **undercloud** for the undercloud

  - **"*"** for all nodes

- Replace **[OTHER 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.

12.6. REMOVING THE OVERCLOUD

To remove the overcloud, run the **openstack overcloud delete** command.

Procedure

1. Delete an existing overcloud:

```
$ source ~/stackrc
(undercloud) $ openstack overcloud delete overcloud
```

2. Confirm that the overcloud is no longer present in the output of the **openstack stack list** command:

```
(undercloud) $ openstack stack list
```

Deletion takes a few minutes.

3. When the deletion completes, follow the standard steps in the deployment scenarios to recreate your overcloud.

CHAPTER 13. CONFIGURING THE OVERCLOUD WITH ANSIBLE

Ansible is the main method to apply the overcloud configuration. This chapter provides information about how to interact with the overcloud Ansible configuration.

Although director generates the Ansible playbooks automatically, it is a good idea to familiarize yourself with Ansible syntax. For more information about using Ansible, see https://docs.ansible.com/.

> **NOTE**
>
> Ansible also uses the concept of roles, which are different to OpenStack Platform director roles. **Ansible roles** form reusable components of playbooks, whereas director roles contain mappings of OpenStack services to node types.

13.1. ANSIBLE-BASED OVERCLOUD CONFIGURATION (CONFIG-DOWNLOAD)

The **config-download** feature is the method that director uses to configure the overcloud. Director uses **config-download** in conjunction with OpenStack Orchestration (heat) and OpenStack Workflow Service (mistral) to generate the software configuration and apply the configuration to each overcloud node. Although heat creates all deployment data from **SoftwareDeployment** resources to perform the overcloud installation and configuration, heat does not apply any of the configuration. Heat only provides the configuration data through the heat API. When director creates the stack, a mistral workflow queries the heat API to obtain the configuration data, generate a set of Ansible playbooks, and applies the Ansible playbooks to the overcloud.

As a result, when you run the **openstack overcloud deploy** command, the following process occurs:

- Director creates a new deployment plan based on **openstack-tripleo-heat-templates** and includes any environment files and parameters to customize the plan.

- Director uses heat to interpret the deployment plan and create the overcloud stack and all descendant resources. This includes provisioning nodes with the OpenStack Bare Metal service (ironic).

- Heat also creates the software configuration from the deployment plan. Director compiles the Ansible playbooks from this software configuration.

- Director generates a temporary user (**tripleo-admin**) on the overcloud nodes specifically for Ansible SSH access.

- Director downloads the heat software configuration and generates a set of Ansible playbooks using heat outputs.

- Director applies the Ansible playbooks to the overcloud nodes using **ansible-playbook**.

13.2. CONFIG-DOWNLOAD WORKING DIRECTORY

Director generates a set of Ansible playbooks for the **config-download** process. These playbooks are stored in a working directory in the **/var/lib/mistral/**. This directory is named after the name of the overcloud, which is **overcloud** by default.

The working directory contains a set of sub-directories named after each overcloud role. These sub-directories contain all tasks relevant to the configuration of the nodes in the overcloud role. These sub-directories also contain additional sub-directories named after each specific node. These sub-

directories contain node-specific variables to apply to the overcloud role tasks. As a result, the overcloud roles within the working directory use the following structure:

```
─ /var/lib/mistral/overcloud
  │
  ├── Controller
  │   ├── overcloud-controller-0
  │   ├── overcloud-controller-1
  │   └── overcloud-controller-2
  ├── Compute
  │   ├── overcloud-compute-0
  │   ├── overcloud-compute-1
  │   └── overcloud-compute-2
  ...
```

Each working directory is a local Git repository that records changes after each deployment operation. Use the local Git repositories to track configuration changes between each deployment.

13.3. ENABLING ACCESS TO CONFIG-DOWNLOAD WORKING DIRECTORIES

The **mistral** user in the OpenStack Workflow service (mistral) containers own all files in the **/var/lib/mistral/** working directories. You can grant the **stack** user on the undercloud access to all files in this directory. This helps with performing certain operations within the directory.

Procedure

1. Use the **setfacl** command to grant the **stack** user on the undercloud access to the files in the **/var/lib/mistral** directory:

   ```
   $ sudo setfacl -R -m u:stack:rwx /var/lib/mistral
   $ sudo chmod -R og-rwx /var/lib/mistral/.ssh
   ```

   This command retains **mistral** user access to the directory.

13.4. CHECKING CONFIG-DOWNLOAD LOG

During the **config-download** process, Ansible creates a log file on the undercloud in the **config-download** working directory.

Procedure

1. View the log with the **less** command within the **config-download** working directory. The following example uses the **overcloud** working directory:

   ```
   $ less /var/lib/mistral/overcloud/ansible.log
   ```

13.5. PERFORMING GIT OPERATIONS ON THE WORKING DIRECTORY

The **config-download** working directory is a local Git repository. Every time a deployment operation runs, director adds a Git commit to the working directory with the relevant changes. You can perform Git operations to view configuration for the deployment at different stages and compare the

configuration with different deployments.

Be aware of the limitations of the working directory. For example, if you use Git to revert to a previous version of the **config-download** working directory, this action affects only the configuration in the working directory. It does not affect the following configurations:

- **The overcloud data schema:**Applying a previous version of the working directory software configuration does not undo data migration and schema changes.

- **The hardware layout of the overcloud:**Reverting to previous software configuration does not undo changes related to overcloud hardware, such as scaling up or down.

- **The heat stack:** Reverting to earlier revisions of the working directory has no effect on the configuration stored in the heat stack. The heat stack creates a new version of the software configuration that applies to the overcloud. To make permanent changes to the overcloud, modify the environment files applied to the overcloud stack before you rerun the **openstack overcloud deploy** command.

Complete the following steps to compare different commits of the **config-download** working directory.

Procedure

1. Change to the **config-download** working directory for your overcloud. In this example, the working directory is for the overcloud named **overcloud**:

   ```
   $ cd /var/lib/mistral/overcloud
   ```

2. Run the **git log** command to list the commits in your working directory. You can also format the log output to show the date:

   ```
   $ git log --format=format:"%h%x09%cd%x09"
   a7e9063 Mon Oct 8 21:17:52 2018 +1000
   dfb9d12 Fri Oct 5 20:23:44 2018 +1000
   d0a910b Wed Oct 3 19:30:16 2018 +1000
   ...
   ```

   By default, the most recent commit appears first.

3. Run the **git diff** command against two commit hashes to see all changes between the deployments:

   ```
   $ git diff a7e9063 dfb9d12
   ```

13.6. DEPLOYMENT METHODS THAT USE CONFIG-DOWNLOAD

There are four main methods that use **config-download** in the context of an overcloud deployment:

Standard deployment

Run the **openstack overcloud deploy** command to automatically run the configuration stage after the provisioning stage. This is the default method when you run the **openstack overcloud deploy** command.

Separate provisioning and configuration

Run the **openstack overcloud deploy** command with specific options to separate the provisioning and configuration stages.

Run the ansible-playbook-command.sh script after a deployment

> Run the **openstack overcloud deploy** command with combined or separate provisioning and configuration stages, then run the **ansible-playbook-command.sh** script supplied in the **config-download** working directory to re-apply the configuration stage.

Provision nodes, manually create config-download, and run Ansible

> Run the **openstack overcloud deploy** command with a specific option to provision nodes, then run the **ansible-playbook** command with the **deploy_steps_playbook.yaml** playbook.

13.7. RUNNING CONFIG-DOWNLOAD ON A STANDARD DEPLOYMENT

The default method for executing **config-download** is to run the **openstack overcloud deploy** command. This method suits most environments.

Prerequisites

- A successful undercloud installation.

- Overcloud nodes ready for deployment.

- Heat environment files that are relevant to your specific overcloud customization.

Procedure

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

2. Source the **stackrc** file:

   ```
   $ source ~/stackrc
   ```

3. Run the deployment command. Include any environment files that you require for your overcloud:

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

4. Wait until the deployment process completes.

During the deployment process, director generates the **config-download** files in a **/var/lib/mistral/** working directory. After the deployment process finishes, view the Ansible playbooks in the working directory to see the tasks director executed to configure the overcloud.

13.8. RUNNING CONFIG-DOWNLOAD WITH SEPARATE PROVISIONING AND CONFIGURATION

The **openstack overcloud deploy** command runs the heat-based provisioning process and then the **config-download** configuration process. You can also run the deployment command to execute each process individually. Use this method to provision your overcloud nodes as a distinct process so that you can perform any manual pre-configuration tasks on the nodes before you run the overcloud configuration process.

Prerequisites

- A successful undercloud installation.

- Overcloud nodes ready for deployment.

- Heat environment files that are relevant to your specific overcloud customization.

Procedure

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

2. Source the **stackrc** file:

   ```
   $ source ~/stackrc
   ```

3. Run the deployment command with the **--stack-only** option. Include any environment files you require for your overcloud:

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

4. Wait until the provisioning process completes.

5. Enable SSH access from the undercloud to the overcloud for the **tripleo-admin** user. The **config-download** process uses the **tripleo-admin** user to perform the Ansible-based configuration:

   ```
   $ openstack overcloud admin authorize
   ```

6. Perform any manual pre-configuration tasks on nodes. If you use Ansible for configuration, use the **tripleo-admin** user to access the nodes.

7. Run the deployment command with the **--config-download-only** option. Include any environment files required for your overcloud:

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

8. Wait until the configuration process completes.

During the configuration stage, director generates the **config-download** files in a **/var/lib/mistral/** working directory. After the deployment process finishes, view the Ansible playbooks in the working directory to see the tasks director executed to configure the overcloud.

13.9. RUNNING CONFIG-DOWNLOAD WITH THE ANSIBLE-PLAYBOOK-COMMAND.SH SCRIPT

When you deploy the overcloud, either with the standard method or a separate provisioning and configuration process, director generates a working directory in **/var/lib/mistral/**. This directory contains the playbooks and scripts necessary to run the configuration process again.

Prerequisites

- An overcloud deployed with the one of the following methods:

  - Standard method that combines provisioning and configuration process

  - Separate provisioning and configuration processes

Procedure

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

2. Change to the directory of the Ansible playbook:

   $ cd /var/lib/mistral/overcloud/

3. Change the owner of **/var/lib/mistral/.ssh** directory to the **stack** user.

   $ sudo chown stack. -R /var/lib/mistral/.ssh/

4. Run the **ansible-playbook-command.sh** command to run the overcloud configuration:

   $ sudo ./ansible-playbook-command.sh

5. Change the owner of **/var/lib/mistral/.ssh** directory to the **mistral** user. This is required to ensure that the ansible-playbook command running inside the **mistral_executor** container is successful.

   $ sudo chown 42430:42430 -R /var/lib/mistral/.ssh/

6. Run the script again as the **mistral** user.
   You can pass additional Ansible arguments to this script, which are then passed unchanged to the **ansible-playbook** command. This means that you can use other Ansible features, such as check mode (**--check**), limiting hosts (**--limit**), or overriding variables (**-e**). For example:

   $ ./ansible-playbook-command.sh --limit Controller

> **⚠ WARNING**
>
> When **--limit** is used to deploy at scale, only hosts included in the execution are added to the SSH **known_hosts** file across the nodes. Therefore, some operations, such as live migration, may not work across nodes that are not in the **known_hosts** file.

7. Wait until the configuration process completes.

Additional information

- The working directory contains a playbook called **deploy_steps_playbook.yaml**, which manages the overcloud configuration tasks. To view this playbook, run the following command:

  ```
  $ less deploy_steps_playbook.yaml
  ```

  The playbook uses various task files contained in the working directory. Some task files are common to all OpenStack Platform roles and some are specific to certain OpenStack Platform roles and servers.

- The working directory also contains sub-directories that correspond to each role that you define in your overcloud **roles_data** file. For example:

  ```
  $ ls Controller/
  ```

  Each OpenStack Platform role directory also contains sub-directories for individual servers of that role type. The directories use the composable role hostname format:

  ```
  $ ls Controller/overcloud-controller-0
  ```

- The Ansible tasks in **deploy_steps_playbook.yaml** are tagged. To see the full list of tags, use the CLI option **--list-tags** with **ansible-playbook**:

  ```
  $ ansible-playbook -i tripleo-ansible-inventory.yaml --list-tags deploy_steps_playbook.yaml
  ```

  Then apply tagged configuration using the **--tags**, **--skip-tags**, or **--start-at-task** with the **ansible-playbook-command.sh** script:

  ```
  $ ./ansible-playbook-command.sh --tags overcloud
  ```

  1. When you run the **config-download** playbooks against the overcloud, you might receive a message regarding the SSH fingerprint for each host. To avoid these messages, include **--ssh-common-args="-o StrictHostKeyChecking=no"** when you run the **ansible-playbook-command.sh** script:

     ```
     $ ./ansible-playbook-command.sh --tags overcloud --ssh-common-args="-o StrictHostKeyChecking=no"
     ```

13.10. RUNNING CONFIG-DOWNLOAD WITH MANUALLY CREATED PLAYBOOKS

You can create your own **config-download** files outside of the standard workflow. For example, you can run the **openstack overcloud deploy** command with the **--stack-only** option to provision the nodes, and then manually apply the Ansible configuration separately.

Prerequisites

- A successful undercloud installation.

- Overcloud nodes ready for deployment.

- Heat environment files that are relevant to your specific overcloud customization.

Procedure

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

2. Source the **stackrc** file:

   ```
   $ source ~/stackrc
   ```

3. Run the deployment command with the **--stack-only** option. Include any environment files required for your overcloud:

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

4. Wait until the provisioning process completes.

5. Enable SSH access from the undercloud to the overcloud for the **tripleo-admin** user. The **config-download** process uses the **tripleo-admin** user to perform the Ansible-based configuration:

   ```
   $ openstack overcloud admin authorize
   ```

6. Generate the **config-download** files:

   ```
   $ openstack overcloud config download \
     --name overcloud \
     --config-dir ~/config-download
   ```

   - **--name** is the name of the overcloud that you want to use for the Ansible file export.

   - **--config-dir** is the location where you want to save the **config-download** files.

7. Change to the directory that contains your **config-download** files:

   ```
   $ cd ~/config-download
   ```

8. Generate a static inventory file:

```
$ tripleo-ansible-inventory \
  --stack <overcloud> \
  --ansible_ssh_user heat-admin \
  --static-yaml-inventory inventory.yaml
```

- Replace **<overcloud>** with the name of your overcloud.

9. Use the **config-download** files and the static inventory file to perform a configuration. To execute the deployment playbook, run the **ansible-playbook** command:

```
$ ansible-playbook \
  -i inventory.yaml \
  -e gather_facts=true \
  -e @global_vars.yaml \
  --private-key ~/.ssh/id_rsa \
  --become \
  ~/config-download/deploy_steps_playbook.yaml
```

10. Wait until the configuration process completes.

11. To generate an **overcloudrc** file manually from this configuration, run the following command:

```
$ openstack action execution run \
  --save-result \
  --run-sync \
  tripleo.deployment.overcloudrc \
  '{"container":"overcloud"}' \
  | jq -r '.["result"]["overcloudrc.v3"]' > overcloudrc.v3
```

12. Manually set the deployment status to success:

```
$ openstack workflow execution create
tripleo.deployment.v1.set_deployment_status_success '{"plan": "<OVERCLOUD>"}'
```

- Replace **<OVERCLOUD>** with the name of your overcloud.

Additional information

- The **config-download** directory contains a playbook called **deploy_steps_playbook.yaml**, which runs the overcloud configuration. To view this playbook, run the following command:

```
$ less deploy_steps_playbook.yaml
```

The playbook uses various task files contained in the working directory. Some task files are common to all OpenStack Platform roles and some are specific to certain OpenStack Platform roles and servers.

- The **config-download** directory also contains sub-directories that correspond to each role that you define in your overcloud **roles_data** file. For example:

```
$ ls Controller/
```

Each OpenStack Platform role directory also contains sub-directories for individual servers of that role type. The directories use the composable role hostname format:

```
$ ls Controller/overcloud-controller-0
```

- The Ansible tasks in **deploy_steps_playbook.yaml** are tagged. To see the full list of tags, use the CLI option **--list-tags** with **ansible-playbook**:

```
$ ansible-playbook -i tripleo-ansible-inventory.yaml --list-tags deploy_steps_playbook.yaml
```

Then apply tagged configuration using the **--tags**, **--skip-tags**, or **--start-at-task** with the **ansible-playbook-command.sh** script:

```
$ ansible-playbook \
  -i inventory.yaml \
  -e gather_facts=true \
  -e @global_vars.yaml \
  --private-key ~/.ssh/id_rsa \
  --become \
  --tags overcloud \
  ~/config-download/deploy_steps_playbook.yaml
```

1. When you run the **config-download** playbooks against the overcloud, you might receive a message regarding the SSH fingerprint for each host. To avoid these messages, include **--ssh-common-args="-o StrictHostKeyChecking=no"** to your **ansible-playbook** command:

```
$ ansible-playbook \
  -i inventory.yaml \
  -e gather_facts=true \
  -e @global_vars.yaml \
  --private-key ~/.ssh/id_rsa \
  --ssh-common-args="-o StrictHostKeyChecking=no" \
  --become \
  --tags overcloud \
  ~/config-download/deploy_steps_playbook.yaml
```

13.11. LIMITATIONS OF CONFIG-DOWNLOAD

The **config-download** feature has some limitations:

- When you use ansible-playbook CLI arguments such as **--tags**, **--skip-tags**, or **--start-at-task**, do not run or apply deployment configuration out of order. These CLI arguments are a convenient way to rerun previously failed tasks or to iterate over an initial deployment. However, to guarantee a consistent deployment, you must run all tasks from **deploy_steps_playbook.yaml** in order.

- You can not use the **--start-at-task** arguments for certain tasks that use a variable in the task name. For example, the **--start-at-task** arguments does not work for the following Ansible task:

```
- name: Run puppet host configuration for step {{ step }}
```

- If your overcloud deployment includes a director-deployed Ceph Storage cluster, you cannot skip **step1** tasks when you use the **--check** option unless you also skip **external_deploy_steps** tasks.

- You can set the number of parallel Ansible tasks with the **--forks** option. However, the performance of **config-download** operations degrades after 25 parallel tasks. For this reason, do not exceed 25 with the **--forks** option.

13.12. CONFIG-DOWNLOAD TOP LEVEL FILES

The following file are important top level files within a **config-download** working directory.

Ansible configuration and execution

The following files are specific to configuring and executing Ansible within the **config-download** working directory.

ansible.cfg

Configuration file used when running **ansible-playbook**.

ansible.log

Log file from the last run of **ansible-playbook**.

ansible-errors.json

JSON structured file that contains any deployment errors.

ansible-playbook-command.sh

Executable script to rerun the **ansible-playbook** command from the last deployment operation.

ssh_private_key

Private SSH key that Ansible uses to access the overcloud nodes.

tripleo-ansible-inventory.yaml

Ansible inventory file that contains hosts and variables for all the overcloud nodes.

overcloud-config.tar.gz

Archive of the working directory.

Playbooks

The following files are playbooks within the **config-download** working directory.

deploy_steps_playbook.yaml

Main deployment steps. This playbook performs the main configuration operations for your overcloud.

pre_upgrade_rolling_steps_playbook.yaml

Pre upgrade steps for major upgrade

upgrade_steps_playbook.yaml

Major upgrade steps.

post_upgrade_steps_playbook.yaml

Post upgrade steps for major upgrade.

update_steps_playbook.yaml

Minor update steps.

fast_forward_upgrade_playbook.yaml

Fast forward upgrade tasks. Use this playbook only when you want to upgrade from one long-life version of Red Hat OpenStack Platform to the next.

13.13. CONFIG-DOWNLOAD TAGS

The playbooks use tagged tasks to control the tasks that they apply to the overcloud. Use tags with the **ansible-playbook** CLI arguments **--tags** or **--skip-tags** to control which tasks to execute. The following list contains information about the tags that are enabled by default:

facts

Fact gathering operations.

common_roles

Ansible roles common to all nodes.

overcloud

All plays for overcloud deployment.

pre_deploy_steps

Deployments that happen before the **deploy_steps** operations.

host_prep_steps

Host preparation steps.

deploy_steps

Deployment steps.

post_deploy_steps

Steps that happen after the **deploy_steps** operations.

external

All external deployment tasks.

external_deploy_steps

External deployment tasks that run on the undercloud only.

13.14. CONFIG-DOWNLOAD DEPLOYMENT STEPS

The **deploy_steps_playbook.yaml** playbook configures the overcloud. This playbook applies all software configuration that is necessary to deploy a full overcloud based on the overcloud deployment plan.

This section contains a summary of the different Ansible plays used within this playbook. The play names in this section are the same names that are used within the playbook and that are displayed in the **ansible-playbook** output. This section also contains information about the Ansible tags that are set on each play.

Gather facts from undercloud

Fact gathering for the undercloud node.
Tags: **facts**

Gather facts from overcloud

Fact gathering for the overcloud nodes.
Tags: **facts**

Load global variables

Loads all variables from **global_vars.yaml**.
Tags: **always**

Common roles for TripleO servers

Applies common Ansible roles to all overcloud nodes, including tripleo-bootstrap for installing bootstrap packages, and tripleo-ssh-known-hosts for configuring ssh known hosts.
Tags: **common_roles**

Overcloud deploy step tasks for step 0

Applies tasks from the deploy_steps_tasks template interface.
Tags: **overcloud**, **deploy_steps**

Server deployments

Applies server-specific heat deployments for configuration such as networking and hieradata. Includes NetworkDeployment, <Role>Deployment, <Role>AllNodesDeployment, etc.
Tags: **overcloud**, **pre_deploy_steps**

Host prep steps

Applies tasks from the host_prep_steps template interface.
Tags: **overcloud**, **host_prep_steps**

External deployment step [1,2,3,4,5]

Applies tasks from the external_deploy_steps_tasks template interface. Ansible runs these tasks only against the undercloud node.
Tags: **external**, **external_deploy_steps**

Overcloud deploy step tasks for [1,2,3,4,5]

Applies tasks from the deploy_steps_tasks template interface.
Tags: **overcloud**, **deploy_steps**

Overcloud common deploy step tasks [1,2,3,4,5]

Applies the common tasks performed at each step, including puppet host configuration, **container-puppet.py**, and paunch (container configuration).
Tags: **overcloud**, **deploy_steps**

Server Post Deployments

Applies server specific heat deployments for configuration performed after the 5-step deployment process.
Tags: **overcloud**, **post_deploy_steps**

External deployment Post Deploy tasks

Applies tasks from the external_post_deploy_steps_tasks template interface. Ansible runs these tasks only against the undercloud node.
Tags: **external**, **external_deploy_steps**

CHAPTER 14. MANAGING CONTAINERS WITH ANSIBLE

> **NOTE**
>
> 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.

Red Hat OpenStack Platform 16.1 uses Paunch to manage containers. However, you can also use the Ansible role **tripleo-container-manage** to perform management operations on your containers. If you want to use the **tripleo-container-manage** role, you must first disable Paunch. With Paunch disabled, director uses the Ansible role automatically, and you can also write custom playbooks to perform specific container management operations:

- Collect the container configuration data that heat generates. The **tripleo-container-manage** role uses this data to orchestrate container deployment.

- Start containers.

- Stop containers.

- Update containers.

- Delete containers.

- Run a container with a specific configuration.

Although director performs container management automatically, you might want to customize a container configuration, or apply a hotfix to a container without redeploying the overcloud.

> **NOTE**
>
> This role supports only Podman container management.

Prerequisites

- A successful undercloud installation. For more information, see Section 4.8, "Installing director".

14.1. ENABLING THE TRIPLEO-CONTAINER-MANAGE ANSIBLE ROLE ON THE UNDERCLOUD

> **NOTE**
>
> 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.

Paunch is the default container management mechanism in Red Hat OpenStack Platform 16.1. However, you can also use the **tripleo-container-manage** Ansible role. If you want to use this role, you must disable Paunch.

Prerequisites

- A host machine with a base operating system and the **python3-tripleoclient** package installed. For more information, see Chapter 3, *Preparing for director installation* .

Procedure

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

2. Set the **undercloud_enable_paunch** parameter to **false** in the **undercloud.conf** file:

   undercloud_enable_paunch: false

3. Run the **openstack undercloud install** command:

   $ openstack undercloud install

14.2. ENABLING THE TRIPLEO-CONTAINER-MANAGE ANSIBLE ROLE ON THE OVERCLOUD

NOTE

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.

Paunch is the default container management mechanism in Red Hat OpenStack Platform 16.1. However, you can also use the **tripleo-container-manage** Ansible role. If you want to use this role, you must disable Paunch.

Prerequisites

- A successful undercloud installation. For more information, see Chapter 4, *Installing director on the undercloud*.

Procedure

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

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

   $ source ~/stackrc

3. Include the **/usr/share/openstack-tripleo-heat-templates/environments/disable-paunch.yaml** file in the overcloud deployment command, along with any other environment files that are relevant for your deployment:

   (undercloud) [stack@director ~]$ openstack overcloud deploy --templates \
     -e /usr/share/openstack-tripleo-heat-templates/environments/disable-paunch.yaml
     -e <other_environment_files>
     ...

14.3. PERFORMING OPERATIONS ON A SINGLE CONTAINER
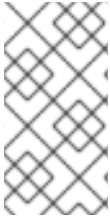
> **NOTE**
>
> This feature is available in this release as a *Technology Preview*, and therefore is not fully supported by Red Hat. It should only be used for testing, and should not be deployed in a production environment. For more information about Technology Preview features, see Scope of Coverage Details.

You can use the **tripleo-container-manage** role to manage all containers, or a specific container. If you want to manage a specific container, you must identify the container deployment step and the name of the container configuration JSON file so that you can target the specific container with a custom Ansible playbook.

Prerequisites

- A successful undercloud installation. For more information, see Chapter 4, *Installing director on the undercloud*.

Procedure

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

2. Source the **overcloudrc** credential file:

   ```
   $ source ~/overcloudrc
   ```

3. Identify the container deployment step. You can find the container configuration for each step in the **/var/lib/tripleo-config/container-startup-config/step_{1,2,3,4,5,6}** directory.

4. Identify the JSON configuration file for the container. You can find the container configuration file in the relevant **step_*** directory. For example, the configuration file for the HAProxy container in step 1 is **/var/lib/tripleo-config/container-startup-config/step_1/haproxy.json**.

5. Write a suitable Ansible playbook. For example, to replace the HAProxy container image, use the following sample playbook:

   ```
   - hosts: localhost
     become: true
     tasks:
       - name: Manage step_1 containers using tripleo-ansible
         block:
           - name: "Manage HAproxy container at step 1 with tripleo-ansible"
             include_role:
               name: tripleo-container-manage
             vars:
               tripleo_container_manage_systemd_order: true
               tripleo_container_manage_config_patterns: 'haproxy.json'
               tripleo_container_manage_config: "/var/lib/tripleo-config/container-startup-
   config/step_1"
               tripleo_container_manage_config_id: "tripleo_step1"
               tripleo_container_manage_config_overrides:
                 haproxy:
                   image: registry.redhat.io/tripleomaster/<HAProxy-container>:hotfix
   ```

For more information about the variables that you can use with the **tripleo-container-manage** role, see Section 14.4, "tripleo-container-manage role variables".

6. Run the playbook:

> (overcloud) [stack@director]$ ansible-playbook <custom_playbook>.yaml

If you want to execute the playbook without applying any changes, include the **--check** option in the **ansible-playbook** command:

> (overcloud) [stack@director]$ ansible-playbook <custom_playbook>.yaml --check

If you want to identify the changes that your playbook makes to your containers without applying the changes, include the **--check** and **--diff** options in the **ansible-playbook** command:

> (overcloud) [stack@director]$ ansible-playbook <custom_playbook>.yaml --check --diff

14.4. TRIPLEO-CONTAINER-MANAGE ROLE VARIABLES

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

The **tripleo-container-manage** Ansible role contains the following variables:

Table 14.1. Role variables

Name	Default value	Description
tripleo_container_manage_check_puppet_config	false	Use this variable if you want Ansible to check Puppet container configurations. Ansible can identify updated container configuration using the configuration hash. If a container has a new configuration from Puppet, set this variable to **true** so that Ansible can detect the new configuration and add the container to the list of containers that Ansible must restart.
tripleo_container_manage_cli	podman	Use this variable to set the command line interface that you want to use to manage containers. The **tripleo-container-manage** role supports only Podman.

Name	Default value	Description
tripleo_container_manage_concurrency	1	Use this variable to set the number of containers that you want to manage concurrently.
tripleo_container_manage_config	/var/lib/tripleo-config/	Use this variable to set the path to the container configuration directory.
tripleo_container_manage_config_id	tripleo	Use this variable to set the ID of a specific configuration step. For example, set this value to **tripleo_step2** to manage containers for step two of the deployment.
tripleo_container_manage_config_patterns	*.json	Use this variable to set the bash regular expression that identifies configuration files in the container configuration directory.
tripleo_container_manage_debug	false	Use this variable to enable or disable debug mode. Run the **tripleo-container-manage** role in debug mode if you want to run a container with a specific one-time configuration, to output the container commands that manage the lifecycle of containers, or to run no-op container management operations for testing and verification purposes.
tripleo_container_manage_healthcheck_disable	false	Use this variable to enable or disable healthchecks.
tripleo_container_manage_log_path	/var/log/containers/stdouts	Use this variable to set the stdout log path for containers.
tripleo_container_manage_systemd_order	false	Use this variable to enable or disable systemd shutdown ordering with Ansible.
tripleo_container_manage_systemd_teardown	true	Use this variable to trigger the cleanup of obsolete containers.

Name	Default value	Description
tripleo_container_manage_config_overrides	{}	Use this variable to override any container configuration. This variable takes a dictionary of values where each key is the container name and the parameters that you want to override, for example, the container image or user. This variable does not write custom overrides to the JSON container configuration files and any new container deployments, updates, or upgrades revert to the content of the JSON configuration file.
tripleo_container_manage_valid_exit_code	[]	Use this variable to check if a container returns an exit code. This value must be a list, for example, **[0,3]**.

CHAPTER 15. USING THE VALIDATION FRAMEWORK

Red Hat OpenStack Platform includes a validation framework that you can use to verify the requirements and functionality of the undercloud and overcloud. The framework includes two types of validations:

- Manual Ansible-based validations, which you execute through the **openstack tripleo validator** command set.

- Automatic in-flight validations, which execute during the deployment process.

You must understand which validations you want to run, and skip validations that are not relevant to your environment. For example, the pre-deployment validation includes a test for TLS-everywhere. If you do not intend to configure your environment for TLS-everywhere, this test fails. Use the **--validation** option in the **openstack tripleo validator run** command to refine the validation according to your environment.

15.1. ANSIBLE-BASED VALIDATIONS

During the installation of Red Hat OpenStack Platform director, director also installs a set of playbooks from the **openstack-tripleo-validations** package. Each playbook contains tests for certain system requirements and a set of groups that define when to run the test:

no-op

Validations that run a *no-op* (no operation) task to verify to workflow functions correctly. These validations run on both the undercloud and overcloud.

prep

Validations that check the hardware configuration of the undercloud node. Run these validation before you run the **openstack undercloud install** command.

openshift-on-openstack

Validations that check that the environment meets the requirements to be able to deploy OpenShift on OpenStack.

pre-introspection

Validations to run before the nodes introspection using Ironic Inspector.

pre-deployment

Validations to run before the **openstack overcloud deploy** command.

post-deployment

Validations to run after the overcloud deployment has finished.

pre-upgrade

Validations to validate your OpenStack deployment before an upgrade.

post-upgrade

Validations to validate your OpenStack deployment after an upgrade.

15.2. LISTING VALIDATIONS

Run the **openstack tripleo validator list** command to list the different types of validations available.

Procedure

1. Source the **stackrc** file.

   ```
   $ source ~/stackrc
   ```

2. Run the **openstack tripleo validator list** command:

   - To list all validations, run the command without any options:

     ```
     $ openstack tripleo validator list
     ```

   - To list validations in a group, run the command with the **--group** option:

     ```
     $ openstack tripleo validator list --group prep
     ```

   > **NOTE**
   >
   > For a full list of options, run **openstack tripleo validator list --help**.

15.3. RUNNING VALIDATIONS

To run a validation or validation group, use the **openstack tripleo validator run** command. To see a full list of options, use the **openstack tripleo validator run --help** command.

Procedure

1. Source the **stackrc** file:

   ```
   $ source ~/stackrc
   ```

2. Create and validate a static inventory file called **inventory.yaml**.

   ```
   $ tripleo-ansible-inventory --static-yaml-inventory inventory.yaml
   $ openstack tripleo validator run  --group pre-introspection -i inventory.yaml
   ```

3. Enter the **openstack tripleo validator run** command:

   - To run a single validation, enter the command with the **--validation** option and the name of the validation. For example, to check the undercloud memory requirements, enter **--validation check-ram**:

     ```
     $ openstack tripleo validator run --validation check-ram
     ```

     If the overcloud uses a plan name that is different to the default **overcloud** name, set the plan name with the **--plan** option:

     ```
     $ openstack tripleo validator run --validation check-ram --plan myovercloud
     ```

     To run multiple specific validations, use the **--validation** option with a comma-separated list of the validations that you want to run. For more information about viewing the list of available validations, see Listing validations.

   - To run all validations in a group, enter the command with the **--group** option:

```
$ openstack tripleo validator run --group prep
```

To view detailed output from a specific validation, run the **openstack tripleo validator show run --full** command against the UUID of the specific validation from the report:

```
$ openstack tripleo validator show run --full <UUID>
```

15.4. VIEWING VALIDATION HISTORY

Director saves the results of each validation after you run a validation or group of validations. View past validation results with the **openstack tripleo validator show history** command.

Prerequisites

- You have run a validation or group of validations.

Procedure

1. Source the **stackrc** file:

   ```
   $ source ~/stackrc
   ```

2. View a list of all validations:

   ```
   $ openstack tripleo validator show history
   ```

   To view history for a specific validation type, run the same command with the **--validation** option:

   ```
   $ openstack tripleo validator show history --validation ntp
   ```

3. View the log for a specific validation UUID with the **openstack tripleo validator show run --full** command:

   ```
   $ openstack tripleo validator show run --full 7380fed4-2ea1-44a1-ab71-aab561b44395
   ```

15.5. VALIDATION FRAMEWORK LOG FORMAT

After you run a validation or group of validations, director saves a JSON-formatted log from each validation in the **/var/logs/validations** directory. You can view the file manually or use the **openstack tripleo validator show run --full** command to display the log for a specific validation UUID.

Each validation log file follows a specific format:

- **<UUID>_<Name>_<Time>**

  UUID

  The Ansible UUID for the validation.

  Name

  The Ansible name for the validation.

  Time

The start date and time for when you ran the validation.

Each validation log contains three main parts:

- plays

- stats

- validation_output

plays

The **plays** section contains information about the tasks that the director performed as part of the validation:

play

A play is a group of tasks. Each **play** section contains information about that particular group of tasks, including the start and end times, the duration, the host groups for the play, and the validation ID and path.

tasks

The individual Ansible tasks that director runs to perform the validation. Each **tasks** section contains a **hosts** section, which contains the action that occurred on each individual host and the results from the execution of the actions. The **tasks** section also contains a **task** section, which contains the duration of the task.

stats

The **stats** section contains a basic summary of the outcome of all tasks on each host, such as the tasks that succeeded and failed.

validation_output

If any tasks failed or caused a warning message during a validation, the **validation_output** contains the output of that failure or warning.

15.6. IN-FLIGHT VALIDATIONS

Red Hat OpenStack Platform includes in-flight validations in the templates of composable services. In-flight validations verify the operational status of services at key steps of the overcloud deployment process.

In-flight validations run automatically as part of the deployment process. Some in-flight validations also use the roles from the **openstack-tripleo-validations** package.

CHAPTER 16. SCALING OVERCLOUD NODES

If you want to add or remove nodes after the creation of the overcloud, you must update the overcloud.

> **WARNING**
>
> Do not use **openstack server delete** to remove nodes from the overcloud. Follow the procedures in this section to remove and replace nodes correctly.

> **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 16.1. 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 17, *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	
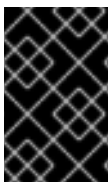
> **IMPORTANT**
>
> Ensure that you have at least 10 GB free space before you scale the overcloud. This free space accommodates image conversion and caching during the node provisioning process.

16.1. ADDING NODES TO THE OVERCLOUD

Complete the following steps to add more nodes to the director node pool.

> **NOTE**
>
> A fresh installation of Red Hat OpenStack Platform 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":"192.168.24.207"
       },
       {
           "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":"192.168.24.208"
       }
     ]
   }
   ```

2. Register the new nodes:

   ```
   $ source ~/stackrc
   $ openstack overcloud node import newnodes.json
   ```

3. After you register the new nodes, launch the introspection process for each new node:

```
$ openstack overcloud node introspect <node_UUID> --provide
```

- Replace **<node_UUID>** with the UUID of the node to add. This process detects and benchmarks the hardware properties of the nodes.

4. Configure the image properties for the node:

```
$ openstack overcloud node configure <node_UUID>
```

16.2. INCREASING NODE COUNTS FOR ROLES

Complete the following steps to scale overcloud nodes for a specific role, such as a Compute node.

Procedure

1. Tag each new node with the role you want. For example, to tag a node with the Compute role, run the following command:

```
$ openstack baremetal node set --property capabilities='profile:compute,boot_option:local' <node_UUID>
```

- Replace **<node_UUID>** with the UUID of the node to tag.

2. To scale the overcloud, you must edit the environment file that contains your node counts and re-deploy the overcloud. For example, to scale your overcloud to 5 Compute nodes, edit the **ComputeCount** parameter:

```
parameter_defaults:
  ...
  ComputeCount: 5
  ...
```

3. Rerun the deployment command with the updated file, which in this example is called **node-info.yaml**:

```
$ openstack overcloud deploy --templates \
  -e /home/stack/templates/node-info.yaml \
  -e [..]
```

Ensure that you include all environment files and options from your initial overcloud creation. This includes the same scale parameters for non-Compute nodes.

4. Wait until the deployment operation completes.

16.3. REMOVING OR REPLACING COMPUTE NODES

In some situations you need to remove a Compute node from the overcloud. For example, you might need to replace a problematic Compute node or remove a group of Compute nodes to scale down your cloud. When you delete a Compute node the node's index is added by default to the blocklist to prevent the index being reused during scale out operations.

You can replace a removed Compute node after you have removed the node from your overcloud deployment.

Prerequisites

- The Compute service is disabled on the nodes that you want to remove to prevent the nodes from scheduling new instances. To confirm that the Compute service is disabled, use the following command to list the compute services:

  ```
  (overcloud)$ openstack compute service list
  ```

  If the Compute service is not disabled then disable the Compture service:

  ```
  (overcloud)$ openstack compute service set <hostname> nova-compute --disable
  ```

- Replace **<hostname>** with the hostname of the Compute node to disable.

  ### TIP

  Use the **--disable-reason** option to add a short explanation on why the service is being disabled. This is useful if you intend to redeploy the Compute service.

- The workloads on the Compute nodes have been migrated to other Compute nodes. For more information, see Migrating virtual machine instances between Compute nodes .

- If Instance HA is enabled, choose one of the following options:

  - If the Compute node is accessible, log in to the Compute node as the **root** user and perform a clean shutdown with the **shutdown -h now** command.

  - If the Compute node is not accessible, log in to a Controller node as the **root** user, disable the STONITH device for the Compute node, and shut down the bare metal node:

    ```
    $ sudo pcs stonith disable <compute_UUID>
    ```

  - Source the **stackrc** undercloud credentials file and power off the baremetal node:

    ```
    $ source ~/stackrc
    (undercloud)$ openstack baremetal node power off <compute_UUID>
    ```

- Replace **<compute_UUID>** with the UUID of the Compute node to remove.

Procedure

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

   ```
   $ source ~/stackrc
   ```

2. Identify the name of the overcloud stack:

   ```
   (undercloud)$ openstack stack list
   ```

3. Identify the UUIDs or hostnames of the Compute nodes that you want to delete:

```
(undercloud)$ openstack server list
```

4. Optional: Run the **overcloud deploy** command with the **--update-plan-only** option to update the plans with the most recent configurations from the templates. This ensures that the overcloud configuration is up-to-date before you delete any Compute nodes:

```
(undercloud)$ openstack overcloud deploy --stack <overcloud> --update-plan-only \
  --templates  \
  -e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml \
  -e /home/stack/templates/network-environment.yaml \
  -e /home/stack/templates/storage-environment.yaml \
  -e /home/stack/templates/rhel-registration/environment-rhel-registration.yaml \
  -e [...]
```

Replace **<overcloud>** with the name of the overcloud stack.
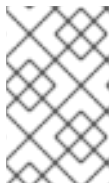
> **NOTE**
>
> You must update the overcloud plans if you updated the overcloud node blocklist. For more information about adding overcloud nodes to the blocklist, see Blocklisting nodes.

5. Delete the Compute nodes from the stack:

```
(undercloud)$ openstack overcloud node delete --stack <overcloud> \
  <node_1> ... [node_n]
```

- Replace **<overcloud>** with the name of the overcloud stack.

- Replace **<node_1>**, and optionally all nodes up to **[node_n]**, with the Compute service hostname or UUID of the Compute nodes you want to delete. Do not use a mix of UUIDs and hostnames. Use either only UUIDs or only hostnames.
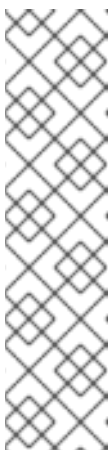
> **NOTE**
>
> If the node has already been powered off, this command returns a **WARNING** message:
>
> > Ansible failed, check log at /var/lib/mistral/overcloud/ansible.log
> > WARNING: Scale-down configuration error. Manual cleanup of some actions may be necessary. Continuing with node removal.
>
> To address the issues caused by the powered off node, nodes manually, complete steps 1 through 8 in Completing the removal of an unreachable Compute node, and then proceed with the next step in this procedure.

6. Wait until the Compute nodes are deleted.

7. Delete the network agents for each node that you deleted:

```
(undercloud)$ source ~/overcloudrc
(overcloud)$ for AGENT in $(openstack network agent list \
  --host <scaled_down_node> -c ID -f value) ; \
```

```
do openstack network agent delete $AGENT ; done
```

Replace **<scaled down node>** with the hostname of the node that you deleted.

8. Check the command output. Because of a bug in RHOSP 16.1.7 and older, you might see a message indicating that the agents could not be deleted.

```
Bad agent request: OVN agents cannot be deleted.
```

If you do not see a **Bad agent request** message, proceed to the next step.

If you see a **Bad agent request** message, go to Deleting the network agents: workaround for bug. After completing the workaround procedure, return here and proceed to the next step.

9. Check the status of the overcloud stack when the node deletion is complete:

```
(overcloud)$ source ~/stackrc
(undercloud)$ openstack stack list
```

Table 16.2. Result

Status	Description
UPDATE_COMPLETE	The Compute node deletion completed successfully. Proceed to the next step.
UPDATE_FAILED	The Compute node deletion failed. A common reason for a failed Compute node deletion is an unreachable IPMI interface on a node that you want to remove. When the deletion fails, you must complete the process manually. Proceed to Completing the removal of an unreachable Compute node to complete the Compute node removal.

10. If Instance HA is enabled, perform the following actions:

    a. Clean up the Pacemaker resources for the Compute node:

    ```
    $ sudo pcs resource delete <compute_UUID>
    $ sudo cibadmin -o nodes --delete --xml-text '<node id="<compute_UUID>"/>'
    $ sudo cibadmin -o fencing-topology --delete --xml-text '<fencing-level target="
    <compute_UUID>"/>'
    $ sudo cibadmin -o status --delete --xml-text '<node_state id="<compute_UUID>"/>'
    $ sudo cibadmin -o status --delete-all --xml-text '<node id="<compute_UUID>"/>' --force
    ```

    b. Delete the STONITH device for the node:

    ```
    $ sudo pcs stonith delete <compute_UUID>
    ```

11. If you are not replacing the removed Compute nodes on the overcloud, then decrease the

ComputeCount parameter in the environment file that contains your node counts. This file is usually named **node-info.yaml**. For example, decrease the node count from four nodes to three nodes if you removed one node:

```
parameter_defaults:
  ...
  ComputeCount: 3
```

Decreasing the node count ensures that director does not provision any new nodes when you run **openstack overcloud deploy**.

If you are replacing the removed Compute node on your overcloud deployment, see Replacing a removed Compute node.

16.3.1. Completing the removal of an unreachable Compute node

If the **openstack overcloud node delete** command failed due to an unreachable node, then you must manually complete the removal of the Compute node from the overcloud.

Prerequisites

- Performing the Removing or replacing a Compute node procedure returned a status of **UPDATE_FAILED**.

Procedure

1. Identify the UUID of the overcloud stack:

   ```
   (undercloud)$ openstack stack list
   ```

2. Identify the UUID of the node that you want to manually delete:

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

3. Set the node that you want to delete to maintenance mode:

   ```
   (undercloud)$ openstack baremetal node maintenance set <UUID>
   ```

   - Replace **<UUID>** with the UUID of the node to put into maintenance mode.

4. Wait for the Compute service to synchronize its state with the Bare Metal service. This can take up to four minutes.

5. Source the overcloud configuration:

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

6. Confirm that the Compute service is disabled on the deleted node on the overcloud, to prevent the node from scheduling new instances:

   ```
   (overcloud)$ openstack compute service list
   ```

   If the Compute service is not disabled then disable it:

```
(overcloud)$ openstack compute service set <hostname> nova-compute --disable
```

- Replace **<hostname>** with the hostname of the Compute node.

  TIP

  Use the **--disable-reason** option to add a short explanation on why the service is being disabled. This is useful if you intend to redeploy the Compute service.

7. Remove the Compute service from the deleted Compute node:

   ```
   (overcloud)$ openstack compute service delete <service_id>
   ```

   - Replace **<service_id>** with the ID of the Compute service that was running on the deleteed node.

8. Remove the deleted Compute service as a resource provider from the Placement service:

   ```
   (overcloud)$ openstack resource provider list
   (overcloud)$ openstack resource provider delete <UUID>
   ```

9. Source the undercloud configuration:

   ```
   (overcloud)$ source ~/stackrc
   ```

10. Delete the Compute node from the stack:

    ```
    (undercloud)$ openstack overcloud node delete --stack <overcloud> <node>
    ```

    - Replace **<overcloud>** with the name or UUID of the overcloud stack.

    - Replace **<node>** with the Compute service hostname or UUID of the Compute node that you want to delete.

      NOTE

      If the node has already been powered off, this command returns a **WARNING** message:

      ```
      Ansible failed, check log at `/var/lib/mistral/overcloud/ansible.log`
      WARNING: Scale-down configuration error. Manual cleanup of some
      actions may be necessary. Continuing with node removal.
      ```

      You can ignore this message.

11. Wait for the overcloud node to be deleted.

12. Source the overcloud configuration:

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

13. Delete the network agents for the node that you deleted:

```
(overcloud)$ for AGENT in $(openstack network agent list \
  --host <scaled_down_node> -c ID -f value) ; \
  do openstack network agent delete $AGENT ; done
```

- Replace **<scaled_down_node>** with the name of the node you deleted.

14. Check the command output. Because of a bug in RHOSP 16.1.7 and older, you might see a message indicating that the agents could not be deleted.

Bad agent request: OVN agents cannot be deleted.

If you do not see this message, proceed to the next step.

If you see this message, complete the procedure in Deleting the network agents: workaround for bug. After completing the workaround procedure, return here and proceed to the next step.

15. Source the undercloud configuration:

```
(overcloud)$ source ~/stackrc
```

16. Check the status of the overcloud stack when the node deletion is complete:

```
(undercloud)$ openstack stack list
```

Table 16.3. Result

Status	Description
UPDATE_COMPLETE	The Compute node deletion completed successfully. Proceed to the next step.
UPDATE_FAILED	The Compute node deletion failed. If the Compute node deletion fails while the node is in maintenance mode, then the problem might be with the hardware. Check the hardware.

17. If Instance HA is enabled, perform the following actions:

    a. Clean up the Pacemaker resources for the node:

    ```
    $ sudo pcs resource delete <scaled_down_node>
    $ sudo cibadmin -o nodes --delete --xml-text '<node id="<scaled_down_node>"/>'
    $ sudo cibadmin -o fencing-topology --delete --xml-text '<fencing-level target="
    <scaled_down_node>"/>'
    $ sudo cibadmin -o status --delete --xml-text '<node_state id="<scaled_down_node>"/>'
    $ sudo cibadmin -o status --delete-all --xml-text '<node id="<scaled_down_node>"/>' --
    force
    ```

    b. Delete the STONITH device for the node:

    ```
    $ sudo pcs stonith delete <device-name>
    ```

18. If you are not replacing the removed Compute node on the overcloud, then decrease the **ComputeCount** parameter in the environment file that contains your node counts. This file is usually named **node-info.yaml**. For example, decrease the node count from four nodes to three nodes if you removed one node:

```
parameter_defaults:
  ...
  ComputeCount: 3
  ...
```

Decreasing the node count ensures that director does not provision any new nodes when you run **openstack overcloud deploy**.

If you are replacing the removed Compute node on your overcloud deployment, see Replacing a removed Compute node.

16.3.2. Deleting the network agents: workaround for bug

After you remove a Compute node, you must delete the associated network agent. If your deployment uses RHOSP 16.1.7 or earlier, a bug prevents you from deleting network agents as expected. See BZ1788336-ovn-controllers are listed as agents but cannot be removed .

With this bug, when you attempt to delete the agent as instructed, the Networking service displays the following error message:

```
Bad agent request: OVN agents cannot be deleted.
```

If you see that error message, perform the following steps to delete the agent.

Prerequisites

- Your attempt to delete network agents after removing a Compute node failed, as indicated by the following error message:

```
Bad agent request: OVN agents cannot be deleted.
```

Procedure

1. List the overcloud nodes:

```
(undercloud)$ openstack server list
```

2. Log into a Controller node as a user with root privileges.

```
$ ssh heat-admin@controller-0.ctlplane
```

3. If you have not done so already, set up command aliases to simplify access to the ovn-sbctl command on the ovn_controller container. For more information, see Creating aliases for OVN troubleshooting commands.

4. Obtain the IP address from the **ovn-controller.log** file:

```
$ sudo less /var/log/containers/openvswitch/ovn-controller.log
```

If **ovn-controller.log** is empty try **ovn-controller.log.1**.

5. Confirm that the IP address is correct:

```
$ ovn-sbctl list encap |grep -a3 <IP_address_from_ovn-controller.log>
```

Replace <IP_address_from_ovn-controller.log> with the IP address from the controller log file.

6. Delete the chassis that contains the IP address:

```
$ ovn-sbctl chassis-del <chassis-name>
```

Replace <chassis-id> with the **chassis_name** value from the output of the **ovn-sbctl list encap** command in the previous step.

7. Check the Chassis_Private table to confirm that chassis has been removed:

```
$ ovn-sbctl find Chassis_private chassis="[]"
```

8. If any chasis are listed, remove each with the following command:

```
$ ovn-sbctl destroy Chassis_Private <listed_name>
```

Replace **<listed_name>** with the name of the chassis to delete.

9. Return to the procedure to complete the removal of the Compute node.

16.3.3. Replacing a removed Compute node

To replace a removed Compute node on your overcloud deployment, you can register and inspect a new Compute node or re-add the removed Compute node. You must also configure your overcloud to provision the node.

Procedure

1. Optional: To reuse the index of the removed Compute node, configure the **RemovalPoliciesMode** and the **RemovalPolicies** parameters for the role to replace the denylist when a Compute node is removed:

```
parameter_defaults:
  <RoleName>RemovalPoliciesMode: update
  <RoleName>RemovalPolicies: [{'resource_list': []}]
```

2. Replace the removed Compute node:

   - To add a new Compute node, register, inspect, and tag the new node to prepare it for provisioning. For more information, see Configuring a basic overcloud .

   - To re-add a Compute node that you removed manually, remove the node from maintenance mode:

     ```
     $ openstack baremetal node maintenance unset <node_uuid>
     ```

3. Rerun the **openstack overcloud deploy** command that you used to deploy the existing overcloud.

4. Wait until the deployment process completes.

5. Confirm that director has successfully registered the new Compute node:

```
$ openstack baremetal node list
```

6. If you performed step 1 to set the **RemovalPoliciesMode** for the role to **update**, then you must reset the **RemovalPoliciesMode** for the role to the default value, **append**, to add the Compute node index to the current denylist when a Compute node is removed:

```
parameter_defaults:
  <RoleName>RemovalPoliciesMode: append
```

7. Rerun the **openstack overcloud deploy** command that you used to deploy the existing overcloud.

16.4. PRESERVING HOSTNAMES WHEN REPLACING NODES THAT USE PREDICTABLE IP ADDRESSES AND HOSTNAMEMAP

If you configured your overcloud to use predictable IP addresses, and **HostNameMap** to map heat-based hostnames to the hostnames of pre-provisioned nodes, then you must configure your overcloud to map the new replacement node index to an IP address and hostname.

Procedure

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

2. Source the **stackrc** file:

```
$ source ~/stackrc
```

3. Retrieve the **physical_resource_id** and the **removed_rsrc_list** for the resource you want to replace:

```
$ openstack stack resource show <stack> <role>
```

- Replace **<stack>** with the name of the stack the resource belongs to, for example, **overcloud**.

- Replace **<role>** with the name of the role that you want to replace the node for, for example, **Compute**.
  Example output:

```
+----------------------+----------------------------------------------------------+
| Field                | Value                                                    |
+----------------------+----------------------------------------------------------+
| attributes           | {u'attributes': None, u'refs': None, u'refs_map': None,  |
|                      | u'removed_rsrc_list': [u'2', u'3']}         | ❶
| creation_time        | 2017-09-05T09:10:42Z                                     |
| description          |                                                          |
```

```
|                |                                                              |
| links          | [{u'href': u'http://192.168.24.1:8004/v1/bd9e6da805594de9 |
|                | 8d4a1d3a3ee874dd/stacks/overcloud/1c7810c4-8a1e-        |
|                | 4d61-a5d8-9f964915d503/resources/Compute', u'rel':      |
|                | u'self'}, {u'href': u'http://192.168.24.1:8004/v1/bd9e6da |
|                | 805594de98d4a1d3a3ee874dd/stacks/overcloud/1c7810c4-8a1e- |
|                | 4d61-a5d8-9f964915d503', u'rel': u'stack'}, {u'href': u'h |
|                | ttp://192.168.24.1:8004/v1/bd9e6da805594de98d4a1d3a3ee874 |
|                | dd/stacks/overcloud-Compute-zkjccox63svg/7632fb0b-       |
|                | 80b1-42b3-9ea7-6114c89adc29', u'rel': u'nested'}]        |
| logical_resource_id   | Compute                                          |
| physical_resource_id  | 7632fb0b-80b1-42b3-9ea7-6114c89adc29             |
| required_by           | [u'AllNodesDeploySteps',                         |
|                       | u'ComputeAllNodesValidationDeployment',          |
|                       | u'AllNodesExtraConfig', u'ComputeIpListMap',      |
|                       | u'ComputeHostsDeployment', u'UpdateWorkflow',    |
|                       | u'ComputeSshKnownHostsDeployment', u'hostsConfig', |
|                       | u'SshKnownHostsConfig', u'ComputeAllNodesDeployment'] |
| resource_name         | Compute                                          |
| resource_status       | CREATE_COMPLETE                                  |
| resource_status_reason | state changed                                   |
| resource_type         | OS::Heat::ResourceGroup                          |
| updated_time          | 2017-09-05T09:10:42Z                             |
+----------------------+---------------------------------------------------------+
```

1   The **removed_rsrc_list** lists the indexes of nodes that have already been removed for the resource.

4. Retrieve the **resource_name** to determine the maximum index that heat has applied to a node for this resource:

   ```
   $ openstack stack resource list <physical_resource_id>
   ```

   - Replace **<physical_resource_id>** with the ID you retrieved in step 3.

5. Use the **resource_name** and the **removed_rsrc_list** to determine the next index that heat will apply to a new node:

   - If **removed_rsrc_list** is empty, then the next index will be (current_maximum_index) + 1.

   - If **removed_rsrc_list** includes the value (current_maximum_index) + 1, then the next index will be the next available index.

6. Retrieve the ID of the replacement bare-metal node:

   ```
   $ openstack baremetal node list
   ```

7. Update the capability of the replacement node with the new index:

   ```
   $ openstack baremetal node set --property capabilities='node:<role>-
   <index>,boot_option:local' <node>
   ```

   - Replace **<role>** with the name of the role that you want to replace the node for, for example, **compute**.

   - Replace **<index>** with the index calculated in step 5.

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

The Compute scheduler uses the node capability to match the node on deployment.

8. Assign a hostname to the new node by adding the index to the **HostnameMap** configuration, for example:

```
parameter_defaults:
  ControllerSchedulerHints:
    'capabilities:node': 'controller-%index%'
  ComputeSchedulerHints:
    'capabilities:node': 'compute-%index%'
  HostnameMap:
    overcloud-controller-0: overcloud-controller-prod-123-0
    overcloud-controller-1: overcloud-controller-prod-456-0    1
    overcloud-controller-2: overcloud-controller-prod-789-0
    overcloud-controller-3: overcloud-controller-prod-456-0    2
    overcloud-compute-0: overcloud-compute-prod-abc-0
    overcloud-compute-3: overcloud-compute-prod-abc-3    3
    overcloud-compute-8: overcloud-compute-prod-abc-3    4
    ....
```

1	Node that you are removing and replacing with the new node.
2	New node.
3	Node that you are removing and replacing with the new node.
4	New node.

NOTE

Do not delete the mapping for the removed node from **HostnameMap**.

9. Add the IP address for the replacement node to the end of each network IP address list in your network IP address mapping file, **ips-from-pool-all.yaml**. In the following example, the IP address for the new index, **overcloud-controller-3**, is added to the end of the IP address list for each **ControllerIPs** network, and is assigned the same IP address as **overcloud-controller-1** because it replaces **overcloud-controller-1**. The IP address for the new index, **overcloud-compute-8**, is also added to the end of the IP address list for each **ComputeIPs** network, and is assigned the same IP address as the index it replaces, **overcloud-compute-3**:

```
parameter_defaults:
  ControllerIPs:
    ...
    internal_api:
      - 192.168.1.10    1
      - 192.168.1.11    2
      - 192.168.1.12    3
      - 192.168.1.11    4
    ...
    storage:
      - 192.168.2.10
```

```
      - 192.168.2.11
      - 192.168.2.12
      - 192.168.2.11
    ...

  ComputeIPs:
    ...
    internal_api:
      - 172.17.0.10
      - 172.17.0.11
      - 172.17.0.11
    ...
    storage:
      - 172.17.0.10
      - 172.17.0.11
      - 172.17.0.11
    ...
```
5
6
7

1 IP address assigned to index 0, host name **overcloud-controller-prod-123-0**.

2 IP address assigned to index 1, host name **overcloud-controller-prod-456-0**. This node is replaced by index 3. **Do not remove this entry.**

3 IP address assigned to index 2, host name **overcloud-controller-prod-789-0**.

4 IP address assigned to index 3, host name **overcloud-controller-prod-456-0**. This is the new node that replaces index 1.

5 IP address assigned to index 0, host name **overcloud-compute-0**.

6 IP address assigned to index 1, host name **overcloud-compute-3**. This node is replaced by index 2. **Do not remove this entry.**

7 IP address assigned to index 2, host name **overcloud-compute-8**. This is the new node that replaces index 1.

16.5. REPLACING CEPH STORAGE NODES

You can use director to replace Ceph Storage nodes in a director-created cluster. For more information, see the Deploying an Overcloud with Containerized Red Hat Ceph guide.

16.6. REPLACING OBJECT STORAGE NODES

Follow the instructions in this section to understand how to replace Object Storage nodes without impact to the integrity of the cluster. This example involves a three-node Object Storage cluster in which you want to replace the node **overcloud-objectstorage-1** node. The goal of the procedure is to add one more node and then remove the **overcloud-objectstorage-1** node. The new node replaces the **overcloud-objectstorage-1** node.

Procedure

1. Increase the Object Storage count using the **ObjectStorageCount** parameter. This parameter is usually located in **node-info.yaml**, which is the environment file that contains your node counts:

```
parameter_defaults:
  ObjectStorageCount: 4
```

The **ObjectStorageCount** parameter defines the quantity of Object Storage nodes in your environment. In this example, scale the quantity of Object Storage nodes from **3** to **4**.

2. Run the deployment command with the updated **ObjectStorageCount** parameter:

```
$ source ~/stackrc
$ openstack overcloud deploy --templates -e node-info.yaml <environment_files>
```

After the deployment command completes, the overcloud contains an additional Object Storage node.

3. Replicate data to the new node. Before you remove a node, in this case, **overcloud-objectstorage-1**, 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 should log entries 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*
```

4. To remove the old node from the ring, reduce the **ObjectStorageCount** parameter to omit the old node. In this example, reduce the **ObjectStorageCount** parameter to **3**:

```
parameter_defaults:
  ObjectStorageCount: 3
```

5. Create a new environment file named **remove-object-node.yaml**. This file identifies and removes the specified Object Storage node. The following content specifies the removal of **overcloud-objectstorage-1**:

```
parameter_defaults:
  ObjectStorageRemovalPolicies:
    [{'resource_list': ['1']}]
```

6. Include both the **node-info.yaml** and **remove-object-node.yaml** files in the deployment command:

```
$ openstack overcloud deploy --templates -e node-info.yaml <environment_files> -e remove-object-node.yaml
```

Director deletes the Object Storage node from the overcloud and updates the rest of the nodes on the overcloud to accommodate the node removal.
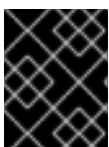
> **IMPORTANT**
>
> Include all environment files and options from your initial overcloud creation. This includes the same scale parameters for non-Compute nodes.

16.7. 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.

16.8. BLOCKLISTING 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 blocklisted nodes are isolated from the effects of the stack operation.

Use the **DeploymentServerBlacklist** parameter in an environment file to create a blocklist.

Setting the blocklist

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
$ openstack overcloud deploy --templates \
  -e server-blocklist.yaml \
  -e [...]
```

Heat blocklists any servers in the list from receiving updated heat deployments. After the stack operation completes, any blocklisted servers remain unchanged. You can also power off or stop the **os-collect-config** agents during the operation.

> **WARNING**
>
> - Exercise caution when you blocklist nodes. Only use a blocklist if you fully understand how to apply the requested change with a blocklist in effect. It is possible to create a hung stack or configure the overcloud incorrectly when you use the blocklist feature. For example, if cluster configuration changes apply to all members of a Pacemaker cluster, blocklisting a Pacemaker cluster member during this change can cause the cluster to fail.
>
> - Do not use the blocklist during update or upgrade procedures. Those procedures have their own methods for isolating changes to particular servers.
>
> - When you add servers to the blocklist, further changes to those nodes are not supported until you remove the server from the blocklist. This includes updates, upgrades, scale up, scale down, and node replacement. For example, when you blocklist existing Compute nodes while scaling out the overcloud with new Compute nodes, the blocklisted 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 blocklisted. 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 Blocklist* section.

Clearing the blocklist

To clear the blocklist 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 17. 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.

17.1. 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
   (undercloud)$ openstack stack list --nested
   ```

   The **overcloud** stack and its subsequent child stacks should have either a **CREATE_COMPLETE** or **UPDATE_COMPLETE**.

2. Install the database client tools:

   ```
   (undercloud)$ sudo dnf -y install mariadb
   ```

3. Configure root user access to the database:

   ```
   (undercloud)$ sudo cp /var/lib/config-data/puppet-generated/mysql/root/.my.cnf /root/.
   ```

4. Perform a backup of the undercloud databases:

   ```
   (undercloud)$ mkdir /home/stack/backup
   (undercloud)$ 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:

   ```
   (undercloud)$ 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:

```
(undercloud)$ 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:

```
(undercloud)$ ssh heat-admin@192.168.0.47 'sudo pcs status'
```

The output shows all services that are running on the existing nodes and 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:

```
(undercloud)$ for i in 192.168.0.46 192.168.0.47 ; do echo "*** $i ***" ; ssh heat-
admin@$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:

```
(undercloud)$ ssh heat-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:

```
(undercloud)$ ssh heat-admin@192.168.0.47 "sudo pcs property show stonith-enabled"
```

Run the following command to disable fencing:

```
(undercloud)$ ssh heat-admin@192.168.0.47 "sudo pcs property set stonith-enabled=false"
```

11. Check the Compute services are active on the director node:

```
(undercloud)$ openstack hypervisor list
```

The output should show all non-maintenance mode nodes as **up**.

12. Ensure all undercloud containers are running:

```
(undercloud)$ sudo podman ps
```

13. Stop all the **nova_*** containers running on the failed Controller node:

```
[root@controller-0 ~]$ sudo systemctl stop tripleo_nova_api.service
[root@controller-0 ~]$ sudo systemctl stop tripleo_nova_api_cron.service
[root@controller-0 ~]$ sudo systemctl stop tripleo_nova_compute.service
[root@controller-0 ~]$ sudo systemctl stop tripleo_nova_conductor.service
[root@controller-0 ~]$ sudo systemctl stop tripleo_nova_metadata.service
[root@controller-0 ~]$ sudo systemctl stop tripleo_nova_placement.service
[root@controller-0 ~]$ sudo systemctl stop tripleo_nova_scheduler.service
```

14. 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.

17.2. 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 and become the root user:

   ```
   # ssh heat-admin@192.168.0.47
   # sudo su -
   ```

   > **NOTE**
   >
   > If the Controller node is unreachable, skip steps 1 and 2 and continue the procedure at step 3 on any working Controller node.

2. Stop the monitor:

   ```
   # systemctl stop ceph-mon@<monitor_hostname>
   ```

   For example:

   ```
   # systemctl stop ceph-mon@overcloud-controller-1
   ```

3. Disconnect from the Controller node that you want to replace.

4. Connect to one of the existing Controller nodes.

```
# ssh heat-admin@192.168.0.46
# sudo su -
```

5. Remove the monitor from the cluster:

```
# sudo podman exec -it ceph-mon-controller-0 ceph mon remove overcloud-controller-1
```

6. On all Controller nodes, remove the v1 and v2 monitor entries from **/etc/ceph/ceph.conf**. For example, if you remove controller-1, then remove the IPs and hostname for controller-1. Before:

```
mon host = [v2:172.18.0.21:3300,v1:172.18.0.21:6789],
[v2:172.18.0.22:3300,v1:172.18.0.22:6789],[v2:172.18.0.24:3300,v1:172.18.0.24:6789]
mon initial members = overcloud-controller-2,overcloud-controller-1,overcloud-controller-0
```

After:

```
mon host = [v2:172.18.0.21:3300,v1:172.18.0.21:6789],
[v2:172.18.0.24:3300,v1:172.18.0.24:6789]
mon initial members = overcloud-controller-2,overcloud-controller-0
```

> **NOTE**
>
> Director updates the **ceph.conf** file on the relevant overcloud nodes when you add the replacement Controller node. Normally, director manages this configuration file exclusively and you should not edit the file manually. However, you can edit the file manually if you want to ensure consistency in case the other nodes restart before you add the new node.

7. (Optional) Archive the monitor data and save the archive on another server:

```
# mv /var/lib/ceph/mon/<cluster>-<daemon_id> /var/lib/ceph/mon/removed-<cluster>-
<daemon_id>
```

17.3. PREPARING THE CLUSTER FOR CONTROLLER NODE REPLACEMENT

Before you replace the old node, you must 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) $ openstack server list -c Name -c Networks
+------------------------+----------------------+
| Name                   | Networks             |
+------------------------+----------------------+
| 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. If the old node is still reachable, log in to one of the remaining nodes and stop pacemaker on the old node. For this example, stop pacemaker on overcloud-controller-1:

```
(undercloud) $ ssh heat-admin@192.168.0.47 "sudo pcs status | grep -w Online | grep -w
overcloud-controller-1"
(undercloud) $ ssh heat-admin@192.168.0.47 "sudo pcs cluster stop overcloud-controller-1"
```

> **NOTE**
>
> In case the old 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 old node, delete the old node from the pacemaker cluster. The following example command logs in to **overcloud-controller-0** to remove **overcloud-controller-1**:

```
(undercloud) $ ssh heat-admin@192.168.0.47 "sudo pcs cluster node remove overcloud-
controller-1"
```

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 heat-admin@192.168.0.47 "sudo pcs cluster node remove overcloud-
controller-1 --skip-offline --force"
```

4. After you remove the old node from the pacemaker cluster, remove the node from the list of known hosts in pacemaker:

```
(undercloud) $ ssh heat-admin@192.168.0.47 "sudo pcs host deauth overcloud-controller-1"
```

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 the replacement, delete the old devices from the node by entering the following command:

```
(undercloud) $ ssh heat-admin@192.168.0.47 "sudo pcs stonith delete
<stonith_resource_name>"
```

- Replace **<stonith_resource_name>** with the name of the STONITH resource that corresponds to the old node. The resource name uses the 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 heat-admin@192.168.0.47 "sudo pcs resource unmanage galera-bundle"
```

17.4. REPLACING A CONTROLLER NODE

To replace a Controller node, identify the index of the node that you want to replace.

- If the node is a virtual node, identify the node that contains the failed disk and restore the disk from a backup. Ensure that the MAC address of the NIC used for PXE boot on the failed server remains the same after disk replacement.

- If the node is a bare metal node, replace the disk, prepare the new disk with your overcloud configuration, and perform a node introspection on the new hardware.

- If the node is a part of a high availability cluster with fencing, you might need recover the Galera nodes separately. For more information, see the article How Galera works and how to rescue Galera clusters in the context of Red Hat OpenStack Platform.

Complete the following example steps to replace the **overcloud-controller-1** node with the **overcloud-controller-3** node. The **overcloud-controller-3** node has the ID **75b25e9a-948d-424a-9b3b-f0ef70a6eacf**.

IMPORTANT

To replace the node with an existing bare metal node, enable maintenance mode on the outgoing node so that the director does not automatically reprovision the node.

Procedure

1. Source the **stackrc** file:

   ```
   $ source ~/stackrc
   ```

2. Identify the index of the **overcloud-controller-1** node:

   ```
   $ INSTANCE=$(openstack server list --name overcloud-controller-1 -f value -c ID)
   ```

3. Identify the bare metal node associated with the instance:

   ```
   $ NODE=$(openstack baremetal node list -f csv --quote minimal | grep $INSTANCE | cut -f1 -d,)
   ```

4. Set the node to maintenance mode:

   ```
   $ openstack baremetal node maintenance set $NODE
   ```

5. If the Controller node is a virtual node, run the following command on the Controller host to replace the virtual disk from a backup:

   ```
   $ cp <VIRTUAL_DISK_BACKUP> /var/lib/libvirt/images/<VIRTUAL_DISK>
   ```

   - Replace **<VIRTUAL_DISK_BACKUP>** with the path to the backup of the failed virtual disk, and replace **<VIRTUAL_DISK>** with the name of the virtual disk that you want to replace. If you do not have a backup of the outgoing node, you must use a new virtualized node.

     If the Controller node is a bare metal node, complete the following steps to replace the disk with a new bare metal disk:

a. Replace the physical hard drive or solid state drive.

b. Prepare the node with the same configuration as the failed node.

6. List unassociated nodes and identify the ID of the new node:

```
$ openstack baremetal node list --unassociated
```

7. Tag the new node with the **control** profile:

```
(undercloud) $ openstack baremetal node set --property
capabilities='profile:control,boot_option:local' 75b25e9a-948d-424a-9b3b-f0ef70a6eacf
```

17.5. 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** section. If the **ExtraConfig** parameter does not exist, 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
```

- Replace **NODE_NAME** with the name of an existing Controller node that you want to use in bootstrap operations after the replacement process.
  If your environment files already include the **ExtraConfig** parameter, add only the lines that set the **pacemaker_short_bootstrap_node_name** and **mysql_short_bootstrap_node_name** parameters.

3. Follow the steps to trigger the Controller node replacement and include the environment files in the **overcloud deploy command**. For more information, see Triggering the Controller node replacement.

For information about troubleshooting the bootstrap Controller node replacement, see the article Replacement of the first Controller node fails at step 1 if the same hostname is used for a new node .

17.6. PRESERVING HOSTNAMES WHEN REPLACING NODES THAT USE PREDICTABLE IP ADDRESSES AND HOSTNAMEMAP

If you configured your overcloud to use predictable IP addresses, and **HostNameMap** to map heat-based hostnames to the hostnames of pre-provisioned nodes, then you must configure your overcloud to map the new replacement node index to an IP address and hostname.

Procedure

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

2. Source the **stackrc** file:

   ```
   $ source ~/stackrc
   ```

3. Retrieve the **physical_resource_id** and the **removed_rsrc_list** for the resource you want to replace:

   ```
   $ openstack stack resource show <stack> <role>
   ```

   - Replace **<stack>** with the name of the stack the resource belongs to, for example, **overcloud**.

   - Replace **<role>** with the name of the role that you want to replace the node for, for example, **Compute**.
     Example output:

     ```
     +-----------------------+-----------------------------------------------------------+
     | Field                 | Value                                                     |
     +-----------------------+-----------------------------------------------------------+
     | attributes            | {u'attributes': None, u'refs': None, u'refs_map': None,   |
     |                       | u'removed_rsrc_list': [u'2', u'3']}          1            |
     | creation_time         | 2017-09-05T09:10:42Z                                      |
     | description           |                                                           |
     | links                 | [{u'href': u'http://192.168.24.1:8004/v1/bd9e6da805594de9 |
     |                       | 8d4a1d3a3ee874dd/stacks/overcloud/1c7810c4-8a1e-          |
     |                       | 4d61-a5d8-9f964915d503/resources/Compute', u'rel':        |
     |                       | u'self'}, {u'href': u'http://192.168.24.1:8004/v1/bd9e6da |
     |                       | 805594de98d4a1d3a3ee874dd/stacks/overcloud/1c7810c4-8a1e- |
     |                       | 4d61-a5d8-9f964915d503', u'rel': u'stack'}, {u'href': u'h |
     |                       | ttp://192.168.24.1:8004/v1/bd9e6da805594de98d4a1d3a3ee874 |
     |                       | dd/stacks/overcloud-Compute-zkjccox63svg/7632fb0b-        |
     |                       | 80b1-42b3-9ea7-6114c89adc29', u'rel': u'nested'}]         |
     | logical_resource_id   | Compute                                                   |
     | physical_resource_id  | 7632fb0b-80b1-42b3-9ea7-6114c89adc29                      |
     | required_by           | [u'AllNodesDeploySteps',                                  |
     |                       | u'ComputeAllNodesValidationDeployment',                   |
     |                       | u'AllNodesExtraConfig', u'ComputeIpListMap',              |
     |                       | u'ComputeHostsDeployment', u'UpdateWorkflow',             |
     |                       | u'ComputeSshKnownHostsDeployment', u'hostsConfig',        |
     |                       | u'SshKnownHostsConfig', u'ComputeAllNodesDeployment']     |
     | resource_name         | Compute                                                   |
     | resource_status       | CREATE_COMPLETE                                           |
     | resource_status_reason | state changed                                            |
     | resource_type         | OS::Heat::ResourceGroup                                   |
     | updated_time          | 2017-09-05T09:10:42Z                                      |
     +-----------------------+-----------------------------------------------------------+
     ```

1. The **removed_rsrc_list** lists the indexes of nodes that have already been removed for the resource.

4. Retrieve the **resource_name** to determine the maximum index that heat has applied to a node for this resource:

   ```
   $ openstack stack resource list <physical_resource_id>
   ```

   - Replace **<physical_resource_id>** with the ID you retrieved in step 3.

5. Use the **resource_name** and the **removed_rsrc_list** to determine the next index that heat will apply to a new node:

   - If **removed_rsrc_list** is empty, then the next index will be (current_maximum_index) + 1.

   - If **removed_rsrc_list** includes the value (current_maximum_index) + 1, then the next index will be the next available index.

6. Retrieve the ID of the replacement bare-metal node:

   ```
   $ openstack baremetal node list
   ```

7. Update the capability of the replacement node with the new index:

   ```
   $ openstack baremetal node set --property capabilities='node:<role>-
   <index>,boot_option:local' <node>
   ```

   - Replace **<role>** with the name of the role that you want to replace the node for, for example, **compute**.

   - Replace **<index>** with the index calculated in step 5.

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

   The Compute scheduler uses the node capability to match the node on deployment.

8. Assign a hostname to the new node by adding the index to the **HostnameMap** configuration, for example:

   ```
   parameter_defaults:
     ControllerSchedulerHints:
       'capabilities:node': 'controller-%index%'
     ComputeSchedulerHints:
       'capabilities:node': 'compute-%index%'
     HostnameMap:
       overcloud-controller-0: overcloud-controller-prod-123-0
       overcloud-controller-1: overcloud-controller-prod-456-0    1
       overcloud-controller-2: overcloud-controller-prod-789-0
       overcloud-controller-3: overcloud-controller-prod-456-0    2
       overcloud-compute-0: overcloud-compute-prod-abc-0
       overcloud-compute-3: overcloud-compute-prod-abc-3    3
       overcloud-compute-8: overcloud-compute-prod-abc-3    4

       ....
   ```

1 Node that you are removing and replacing with the new node.

2 New node.

3 Node that you are removing and replacing with the new node.

4 New node.

> **NOTE**
>
> Do not delete the mapping for the removed node from **HostnameMap**.

9. Add the IP address for the replacement node to the end of each network IP address list in your network IP address mapping file, **ips-from-pool-all.yaml**. In the following example, the IP address for the new index, **overcloud-controller-3**, is added to the end of the IP address list for each **ControllerIPs** network, and is assigned the same IP address as **overcloud-controller-1** because it replaces **overcloud-controller-1**. The IP address for the new index, **overcloud-compute-8**, is also added to the end of the IP address list for each **ComputeIPs** network, and is assigned the same IP address as the index it replaces, **overcloud-compute-3**:

```
parameter_defaults:
  ControllerIPs:
    ...
    internal_api:
      - 192.168.1.10    1
      - 192.168.1.11    2
      - 192.168.1.12    3
      - 192.168.1.11    4
    ...
    storage:
      - 192.168.2.10
      - 192.168.2.11
      - 192.168.2.12
      - 192.168.2.11
    ...

  ComputeIPs:
    ...
    internal_api:
      - 172.17.0.10    5
      - 172.17.0.11    6
      - 172.17.0.11    7
    ...
    storage:
      - 172.17.0.10
      - 172.17.0.11
      - 172.17.0.11
    ...
```

1 IP address assigned to index 0, host name **overcloud-controller-prod-123-0**.

2 IP address assigned to index 1, host name **overcloud-controller-prod-456-0**. This node is replaced by index 3. **Do not remove this entry.**

3 IP address assigned to index 2, host name **overcloud-controller-prod-789-0**.

4 IP address assigned to index 3, host name **overcloud-controller-prod-456-0**. This is the new node that replaces index 1.

5 IP address assigned to index 0, host name **overcloud-compute-0**.

6 IP address assigned to index 1, host name **overcloud-compute-3**. This node is replaced by index 2. **Do not remove this entry.**

7 IP address assigned to index 2, host name **overcloud-compute-8**. This is the new node that replaces index 1.

17.7. TRIGGERING THE CONTROLLER NODE REPLACEMENT

Complete the following steps to remove the old Controller node and replace it with a new Controller node.

Procedure

1. Determine the UUID of the Controller node that you want to remove and store it in the **<NODEID>** variable. Ensure that you replace **<node_name>** with the name of the node that you want to remove:

   ```
   (undercloud)[stack@director ~]$ NODEID=$(openstack server list -f value -c ID --name <node_name>)
   ```

2. To identify the Heat resource ID, enter the following command:

   ```
   (undercloud)[stack@director ~]$ openstack stack resource show overcloud ControllerServers -f json -c attributes | jq --arg NODEID "$NODEID" -c '.attributes.value | keys[] as $k | if .[$k] == $NODEID then "Node index \($k) for \(.[$k])" else empty end'
   ```

3. Create the following environment file ~/**templates**/**remove-controller.yaml** and include the node index of the Controller node that you want to remove:

   ```
   parameters:
     ControllerRemovalPolicies:
       [{'resource_list': ['<node_index>']}]
   ```

4. Enter the overcloud deployment command, and include the **remove-controller.yaml** environment file and any other environment files relevant to your environment:

   ```
   (undercloud) $ openstack overcloud deploy --templates \
     -e /home/stack/templates/remove-controller.yaml \
     [OTHER OPTIONS]
   ```

NOTE

- Include **-e ~/templates/remove-controller.yaml** only for this instance of the deployment command. Remove this environment file from subsequent deployment operations.

- Include **~/templates/bootstrap-controller.yaml** if you are replacing a bootstrap Controller node and want to keep the node name. For more information, see Replacing a bootstrap Controller node .

5. Director removes the old node, creates a new node, and updates the overcloud stack. You can check the status of the overcloud stack with the following command:

```
(undercloud)$ openstack stack list --nested
```

6. When the deployment command completes, confirm that the old node is replaced with the new node:

```
(undercloud) $ openstack server list -c Name -c Networks
+-----------------------+----------------------+
| Name                  | Networks             |
+-----------------------+----------------------+
| overcloud-compute-0    | ctlplane=192.168.0.44 |
| overcloud-controller-0 | ctlplane=192.168.0.47 |
| overcloud-controller-2 | ctlplane=192.168.0.46 |
| overcloud-controller-3 | ctlplane=192.168.0.48 |
+-----------------------+----------------------+
```

The new node now hosts running control plane services.

17.8. CLEANING UP AFTER CONTROLLER NODE REPLACEMENT

After you complete the node replacement, complete the following steps to 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:

```
[heat-admin@overcloud-controller-0 ~]$ sudo pcs resource refresh galera-bundle
[heat-admin@overcloud-controller-0 ~]$ sudo pcs resource manage galera-bundle
```

3. Perform a final status check to ensure that the services are running correctly:

```
[heat-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.

4. Exit to director:

   > [heat-admin@overcloud-controller-0 ~]$ exit

5. Source the **overcloudrc** file so that you can interact with the overcloud:

   > $ source ~/overcloudrc

6. Check the network agents in your overcloud environment:

   > (overcloud) $ openstack network agent list

7. 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

8. 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

9. 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:

      > [heat-admin@overcloud-controller-0 ~]$ sudo podman exec -it cinder_api cinder-manage
      > service remove cinder-backup <host>
      > [heat-admin@overcloud-controller-0 ~]$ sudo podman exec -it cinder_api cinder-manage
      > service remove cinder-scheduler <host>

10. 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:

       > [heat-admin@overcloud-controller-0 ~]$ podman exec -it rabbitmq-bundle-podman-0
       > bash
       > [heat-admin@overcloud-controller-0 ~]$ rabbitmqctl cluster_status

    c. Use the **rabbitmqctl** command to forget the replaced controller node:

       > [heat-admin@overcloud-controller-0 ~]$ rabbitmqctl forget_cluster_node <node_name>

11. 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 .

CHAPTER 18. REBOOTING NODES

You might need to reboot the nodes in the undercloud and overcloud. 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.

18.1. 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.

18.2. 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:

   ```
   [heat-admin@overcloud-controller-0 ~]$ sudo pcs cluster stop
   ```

3. Reboot the node:

```
[heat-admin@overcloud-controller-0 ~]$ sudo reboot
```

4. Wait until the node boots.

Verfication

1. Verify that the services are enabled.

   a. If the node uses Pacemaker services, check that the node has rejoined the cluster:

   ```
   [heat-admin@overcloud-controller-0 ~]$ sudo pcs status
   ```

   b. If the node uses Systemd services, check that all services are enabled:

   ```
   [heat-admin@overcloud-controller-0 ~]$ sudo systemctl status
   ```

   c. If the node uses containerized services, check that all containers on the node are active:

   ```
   [heat-admin@overcloud-controller-0 ~]$ sudo podman ps
   ```

18.3. 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.

18.4. REBOOTING A CEPH STORAGE (OSD) CLUSTER

Complete the following steps to reboot a cluster of Ceph Storage (OSD) nodes.

Procedure

1. Log in to a Ceph MON or Controller node and disable Ceph Storage cluster rebalancing temporarily:

   ```
   $ sudo podman exec -it ceph-mon-controller-0 ceph osd set noout
   $ sudo podman exec -it ceph-mon-controller-0 ceph osd set norebalance
   ```
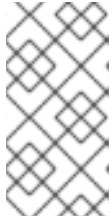
> **NOTE**
>
> If you have a multistack or distributed compute node (DCN) architecture, you must specify the cluster name when you set the **noout** and **norebalance** flags. For example: **sudo podman exec -it ceph-mon-controller-0 ceph osd set noout --cluster <cluster_name>**

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 cluster status:

   ```
   $ sudo podman exec -it ceph-mon-controller-0 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 MON or Controller node and re-enable cluster rebalancing:

   ```
   $ sudo podman exec -it ceph-mon-controller-0 ceph osd unset noout
   $ sudo podman exec -it ceph-mon-controller-0 ceph osd unset norebalance
   ```
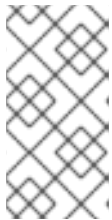
> **NOTE**
>
> If you have a multistack or distributed compute node (DCN) architecture, you must specify the cluster name when you unset the **noout** and **norebalance** flags. For example: **sudo podman exec -it ceph-mon-controller-0 ceph osd set noout --cluster <cluster_name>**

8. Perform a final status check to verify that the cluster reports **HEALTH_OK**:

   ```
   $ sudo podman exec -it ceph-mon-controller-0 ceph status
   ```

18.5. 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.

18.6. REBOOTING COMPUTE NODES

Complete the following steps to reboot Compute nodes. To ensure minimal downtime of instances in your Red Hat OpenStack Platform environment, this procedure also includes instructions about migrating instances from the Compute node that you want to reboot. This involves the following workflow:

- Decide whether to migrate instances to another Compute node before rebooting the node.

- Select and disable the Compute node you want to reboot so that it does not provision new instances.

- Migrate the instances to another Compute node.

- Reboot the empty Compute node.

- 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 run into when migrating virtual machine instances between Compute nodes. For more information, see Migration constraints 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. 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**. 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. List all Compute nodes and their UUIDs:

   ```
   $ source ~/stackrc
   (undercloud) $ openstack server list --name compute
   ```

   Identify the UUID of the Compute node that you want to reboot.

3. From the undercloud, select a Compute node. Disable the node:

```
$ source ~/overcloudrc
(overcloud) $ openstack compute service list
(overcloud) $ openstack compute service set <hostname> nova-compute --disable
```

4. List all instances on the Compute node:

```
(overcloud) $ openstack server list --host <hostname> --all-projects
```

5. If you decide not to migrate instances, skip to this step.

6. If you decide to migrate the instances to another Compute node, use one of the following commands:

   - Migrate the instance to a different host:

   ```
   (overcloud) $ openstack server migrate <instance_id> --live <target_host> --wait
   ```

   - 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.

7. Wait until migration completes.

8. Confirm that the migration was successful:

```
(overcloud) $ openstack server list --host <hostname> --all-projects
```

9. Continue to migrate instances until none remain on the chosen Compute node.

10. Log in to the Compute node and reboot the node:

```
[heat-admin@overcloud-compute-0 ~]$ sudo reboot
```

11. Wait until the node boots.

12. Re-enable the Compute node:

```
$ source ~/overcloudrc
(overcloud) $ openstack compute service set <hostname>  nova-compute --enable
```

13. Check that the Compute node is enabled:

```
(overcloud) $ openstack compute service list
```

CHAPTER 19. 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.

Prerequisites

- A running undercloud and overcloud

19.1. 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

19.2. 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.

19.3. 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.

19.4. 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
```

19.5. 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?"

19.6. 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.

19.7. 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
```

19.8. 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.

19.9. 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

19.10. 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

- A powered down undercloud

Procedure

1. Power on the undercloud and wait until the undercloud boots.

Verification

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

2. Check the services on the undercloud:

   ```
   $ systemctl list-units 'tripleo_*'
   ```

3. Source the credential file for your undercloud and run the validation command to check that all services and containers are active and healthy.

   ```
   $ source stackrc
   $ openstack tripleo validator run --validation service-status --limit undercloud
   ```

Additional resources

- Using the validation framework

19.11. 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

- Powered down Controller nodes

Procedure

1. 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
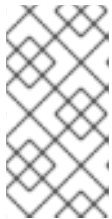
> **NOTE**
>
> If your environment uses Instance HA, the Pacemaker resources do not start until you start the Compute nodes or perform a manual unfence operation with the **pcs stonith confirm <compute_node>** command. You must run this command on each Compute node that uses Instance HA.

19.12. 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?"

19.13. 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
```

19.14. 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 20. CONFIGURING CUSTOM SSL/TLS CERTIFICATES

You can manually configure the undercloud to use SSL/TLS for communication over public endpoints. When you manually configure undercloud endpoints with SSL/TLS, you are creating secure endpoints as a proof-of-concept. Red Hat recommends using a certificate authority solution.

When you use a certificate authority (CA) solution, you have production ready solutions such as a certificate renewals, certificate revocation lists (CRLs), and industry accepted cryptography. For information on using Red Hat Identity Manager (IdM) as a CA, see Implementing TLS-e with Ansible .

If want to you use a SSL certificate with your own certificate authority, you must complete the following configuration steps.

20.1. INITIALIZING THE SIGNING HOST

The signing host is the host that generates and signs new certificates with a certificate authority. If you have never created SSL certificates on the chosen signing host, you might need to initialize the host so that it can sign new certificates.

Procedure

1. The **/etc/pki/CA/index.txt** file contains records of all signed certificates. Ensure that the filesystem path and **index.txt** file are present:

   ```
   $ sudo mkdir -p /etc/pki/CA
   $ sudo touch /etc/pki/CA/index.txt
   ```

2. The **/etc/pki/CA/serial** file identifies the next serial number to use for the next certificate to sign. Check if this file exists. If the file does not exist, create a new file with a new starting value:

   ```
   $ echo '1000' | sudo tee /etc/pki/CA/serial
   ```

20.2. CREATING A CERTIFICATE AUTHORITY

Normally you sign your SSL/TLS certificates with an external certificate authority. In some situations, you might want to use your own certificate authority. For example, you might want to have an internal-only certificate authority.

Procedure

1. Generate a key and certificate pair to act as the certificate authority:

   ```
   $ openssl genrsa -out ca.key.pem 4096
   $ openssl req  -key ca.key.pem -new -x509 -days 7300 -extensions v3_ca -out ca.crt.pem
   ```

2. The **openssl req** command requests certain details about your authority. Enter these details at the prompt. These commands create a certificate authority file called **ca.crt.pem**.

3. Set the certificate location as the value for the **PublicTLSCAFile** parameter in the **enable-tls.yaml** file. When you set the certificate location as the value for the **PublicTLSCAFile** parameter, you ensure that the CA certificate path is added to the **clouds.yaml** authentication

file.

```
parameter_defaults:
    PublicTLSCAFile: /etc/pki/ca-trust/source/anchors/cacert.pem
```

20.3. ADDING THE CERTIFICATE AUTHORITY TO CLIENTS

For any external clients aiming to communicate using SSL/TLS, copy the certificate authority file to each client that requires access to your Red Hat OpenStack Platform environment.

Procedure

1. Copy the certificate authority to the client system:

   ```
   $ sudo cp ca.crt.pem /etc/pki/ca-trust/source/anchors/
   ```

2. After you copy the certificate authority file to each client, run the following command on each client to add the certificate to the certificate authority trust bundle:

   ```
   $ sudo update-ca-trust extract
   ```

20.4. CREATING AN SSL/TLS KEY

Enabling SSL/TLS on an OpenStack environment requires an SSL/TLS key to generate your certificates.

Procedure

1. Run the following command to generate the SSL/TLS key (**server.key.pem**):

   ```
   $ openssl genrsa -out server.key.pem 2048
   ```

20.5. CREATING AN SSL/TLS CERTIFICATE SIGNING REQUEST

Complete the following steps to create a certificate signing request.

Procedure

1. Copy the default OpenSSL configuration file:

   ```
   $ cp /etc/pki/tls/openssl.cnf .
   ```

2. Edit the new **openssl.cnf** file and configure the SSL parameters that you want to use for director. An example of the types of parameters to modify include:

   ```
   [req]
   distinguished_name = req_distinguished_name
   req_extensions = v3_req

   [req_distinguished_name]
   countryName = Country Name (2 letter code)
   ```

```
countryName_default = AU
stateOrProvinceName = State or Province Name (full name)
stateOrProvinceName_default = Queensland
localityName = Locality Name (eg, city)
localityName_default = Brisbane
organizationalUnitName = Organizational Unit Name (eg, section)
organizationalUnitName_default = Red Hat
commonName = Common Name
commonName_default = 192.168.0.1
commonName_max = 64

[ v3_req ]
# Extensions to add to a certificate request
basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
subjectAltName = @alt_names

[alt_names]
IP.1 = 192.168.0.1
DNS.1 = instack.localdomain
DNS.2 = vip.localdomain
DNS.3 = 192.168.0.1
```

Set the **commonName_default** to one of the following entries:

- If you are using an IP address to access director over SSL/TLS, use the **undercloud_public_host** parameter in the **undercloud.conf** file.

- If you are using a fully qualified domain name to access director over SSL/TLS, use the domain name.

Edit the **alt_names** section to include the following entries:

- **IP** – A list of IP addresses that clients use to access director over SSL.

- **DNS** – A list of domain names that clients use to access director over SSL. Also include the Public API IP address as a DNS entry at the end of the **alt_names** section.

> **NOTE**
>
> For more information about **openssl.cnf**, run the **man openssl.cnf** command.

3. Run the following command to generate a certificate signing request (**server.csr.pem**):

   ```
   $ openssl req -config openssl.cnf -key server.key.pem -new -out server.csr.pem
   ```

   Ensure that you include your OpenStack SSL/TLS key with the **-key** option.

This command generates a **server.csr.pem** file, which is the certificate signing request. Use this file to create your OpenStack SSL/TLS certificate.

20.6. CREATING THE SSL/TLS CERTIFICATE

To generate the SSL/TLS certificate for your OpenStack environment, the following files must be present:

openssl.cnf

The customized configuration file that specifies the v3 extensions.

server.csr.pem

The certificate signing request to generate and sign the certificate with a certificate authority.

ca.crt.pem

The certificate authority, which signs the certificate.

ca.key.pem

The certificate authority private key.

Procedure

1. Run the following command to create a certificate for your undercloud or overcloud:

   ```
   $ sudo openssl ca -config openssl.cnf -extensions v3_req -days 3650 -in server.csr.pem -out
   server.crt.pem -cert ca.crt.pem -keyfile ca.key.pem
   ```

   This command uses the following options:

   -config

   Use a custom configuration file, which is the **openssl.cnf** file with v3 extensions.

   -extensions v3_req

   Enabled v3 extensions.

   -days

   Defines how long in days until the certificate expires.

   -in'

   The certificate signing request.

   -out

   The resulting signed certificate.

   -cert

   The certificate authority file.

   -keyfile

   The certificate authority private key.

This command creates a new certificate named **server.crt.pem**. Use this certificate in conjunction with your OpenStack SSL/TLS key

20.7. ADDING THE CERTIFICATE TO THE UNDERCLOUD

Complete the following steps to add your OpenStack SSL/TLS certificate to the undercloud trust bundle.

Procedure

1. Run the following command to combine the certificate and key:

   ```
   $ cat server.crt.pem server.key.pem > undercloud.pem
   ```

   This command creates a **undercloud.pem** file.

2. Copy the **undercloud.pem** file to a location within your **/etc/pki** directory and set the necessary SELinux context so that HAProxy can read it:

```
$ sudo mkdir /etc/pki/undercloud-certs
$ sudo cp ~/undercloud.pem /etc/pki/undercloud-certs/.
$ sudo semanage fcontext -a -t etc_t "/etc/pki/undercloud-certs(/.*)?"
$ sudo restorecon -R /etc/pki/undercloud-certs
```

3. Add the **undercloud.pem** file location to the **undercloud_service_certificate** option in the **undercloud.conf** file:

```
undercloud_service_certificate = /etc/pki/undercloud-certs/undercloud.pem
```

Do not set or enable the **generate_service_certificate** and **certificate_generation_ca** parameters. Director uses these parameters to automatically generate a certificate instead of using the **undercloud.pem** certificate you created manually.

4. Add the certificate authority that signed the certificate to the list of trusted Certificate Authorities on the undercloud so that different services within the undercloud have access to the certificate authority:

```
$ sudo cp ca.crt.pem /etc/pki/ca-trust/source/anchors/
$ sudo update-ca-trust extract
```

To verify the certificate authority was added to the undercloud, use **openssl** to check the trust bundle:

```
$ openssl crl2pkcs7 -nocrl -certfile /etc/pki/tls/certs/ca-bundle.crt | openssl pkcs7 -print_certs -text | grep <CN of the CA issuer> -A 10 -B 10
```

- Replace **<CN of the CA issuer>** with the common name of the issuer of the CA. This command outputs the main certificate details, including the validity dates.

CHAPTER 21. ADDITIONAL INTROSPECTION OPERATIONS

In some situations, you might want to perform introspection outside of the standard overcloud deployment workflow. For example, you might want to introspect new nodes or refresh introspection data after replacing hardware on existing unused nodes.

21.1. PERFORMING INDIVIDUAL NODE INTROSPECTION

To perform a single introspection on an available node, set the node to management mode and perform the introspection.

Procedure

1. Set all nodes to a **manageable** state:

   (undercloud) $ openstack baremetal node manage [NODE UUID]

2. Perform the introspection:

   (undercloud) $ openstack overcloud node introspect [NODE UUID] --provide

   After the introspection completes, the node changes to an **available** state.

21.2. PERFORMING NODE INTROSPECTION AFTER INITIAL INTROSPECTION

After an initial introspection, all nodes enter an **available** state due to the **--provide** option. To perform introspection on all nodes after the initial introspection, set the node to management mode and perform the introspection.

Procedure

1. Set all nodes to a **manageable** state

   (undercloud) $ for node in $(openstack baremetal node list --fields uuid -f value) ; do openstack baremetal node manage $node ; done

2. Run the bulk introspection command:

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

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

21.3. PERFORMING NETWORK INTROSPECTION FOR INTERFACE INFORMATION

Network introspection retrieves link layer discovery protocol (LLDP) data from network switches. The following commands show a subset of LLDP information for all interfaces on a node, or full information for a particular node and interface. This can be useful for troubleshooting. Director enables LLDP data collection by default.

Procedure

1. To get a list of interfaces on a node, run the following command:

   (undercloud) $ openstack baremetal introspection interface list [NODE UUID]

   For example:

   ```
   (undercloud) $ openstack baremetal introspection interface list c89397b7-a326-41a0-907d-
   79f8b86c7cd9
   +-----------+-------------------+----------------------+-----------------+---------------+
   | Interface | MAC Address       | Switch Port VLAN IDs   | Switch Chassis ID | Switch Port ID |
   +-----------+-------------------+----------------------+-----------------+---------------+
   | p2p2      | 00:0a:f7:79:93:19 | [103, 102, 18, 20, 42] | 64:64:9b:31:12:00 | 510          |
   | p2p1      | 00:0a:f7:79:93:18 | [101]                 | 64:64:9b:31:12:00 | 507          |
   | em1       | c8:1f:66:c7:e8:2f | [162]                 | 08:81:f4:a6:b3:80 | 515          |
   | em2       | c8:1f:66:c7:e8:30 | [182, 183]            | 08:81:f4:a6:b3:80 | 559          |
   +-----------+-------------------+----------------------+-----------------+---------------+
   ```

2. To view interface data and switch port information, run the following command:

   (undercloud) $ openstack baremetal introspection interface show [NODE UUID]
   [INTERFACE]

   For example:

   ```
   (undercloud) $ openstack baremetal introspection interface show c89397b7-a326-41a0-
   907d-79f8b86c7cd9 p2p1
   +--------------------------------+--------------------------------------------------------------------
   -------------------------------------------------+
   | Field                          | Value
   |
   +--------------------------------+--------------------------------------------------------------------
   -------------------------------------------------+
   | interface                      | p2p1
   |
   | mac                            | 00:0a:f7:79:93:18
   |
   | node_ident                     | c89397b7-a326-41a0-907d-79f8b86c7cd9
   |
   | switch_capabilities_enabled    | [u'Bridge', u'Router']
   |
   | switch_capabilities_support    | [u'Bridge', u'Router']
   |
   | switch_chassis_id              | 64:64:9b:31:12:00
   |
   | switch_port_autonegotiation_enabled  | True
   |
   | switch_port_autonegotiation_support  | True
   |
   | switch_port_description        | ge-0/0/2.0
   |
   | switch_port_id                 | 507
   |
   | switch_port_link_aggregation_enabled | False
   ```

```
|
| switch_port_link_aggregation_id      | 0
|
| switch_port_link_aggregation_support | True
|
| switch_port_management_vlan_id       | None
|
| switch_port_mau_type              | Unknown
|
| switch_port_mtu               | 1514
|
| switch_port_physical_capabilities    | [u'1000BASE-T fdx', u'100BASE-TX fdx', u'100BASE-
TX hdx', u'10BASE-T fdx', u'10BASE-T hdx', u'Asym and Sym PAUSE fdx'] |
| switch_port_protocol_vlan_enabled    | None
|
| switch_port_protocol_vlan_ids       | None
|
| switch_port_protocol_vlan_support    | None
|
| switch_port_untagged_vlan_id        | 101
|
| switch_port_vlan_ids              | [101]
|
| switch_port_vlans               | [{u'name': u'RHOS13-PXE', u'id': 101}]
|
| switch_protocol_identities         | None
|
| switch_system_name               | rhos-compute-node-sw1
|
+-----------------------------------+-------------------------------------------------------------------
-----------------------------------------------+
```

21.4. RETRIEVING HARDWARE INTROSPECTION DETAILS

The Bare Metal service hardware-inspection-extras feature is enabled by default, and you can use it to retrieve hardware details for overcloud configuration. For more information about the **inspection_extras** parameter in the **undercloud.conf** file, see Configuring the Director.

For example, the **numa_topology** collector is part of the hardware-inspection extras and includes the following information for each NUMA node:

- RAM (in kilobytes)

- Physical CPU cores and their sibling threads

- NICs associated with the NUMA node

Procedure

- To retrieve the information listed above, substitute <UUID> with the UUID of the bare-metal node to complete the following command:

  ```
  # openstack baremetal introspection data save <UUID> | jq .numa_topology
  ```

The following example shows the retrieved NUMA information for a bare-metal node:

```
{
  "cpus": [
    {
      "cpu": 1,
      "thread_siblings": [
        1,
        17
      ],
      "numa_node": 0
    },
    {
      "cpu": 2,
      "thread_siblings": [
        10,
        26
      ],
      "numa_node": 1
    },
    {
      "cpu": 0,
      "thread_siblings": [
        0,
        16
      ],
      "numa_node": 0
    },
    {
      "cpu": 5,
      "thread_siblings": [
        13,
        29
      ],
      "numa_node": 1
    },
    {
      "cpu": 7,
      "thread_siblings": [
        15,
        31
      ],
      "numa_node": 1
    },
    {
      "cpu": 7,
      "thread_siblings": [
        7,
        23
      ],
      "numa_node": 0
    },
    {
      "cpu": 1,
      "thread_siblings": [
        9,
        25
      ],
```

```
      "numa_node": 1
    },
    {
      "cpu": 6,
      "thread_siblings": [
        6,
        22
      ],
      "numa_node": 0
    },
    {
      "cpu": 3,
      "thread_siblings": [
        11,
        27
      ],
      "numa_node": 1
    },
    {
      "cpu": 5,
      "thread_siblings": [
        5,
        21
      ],
      "numa_node": 0
    },
    {
      "cpu": 4,
      "thread_siblings": [
        12,
        28
      ],
      "numa_node": 1
    },
    {
      "cpu": 4,
      "thread_siblings": [
        4,
        20
      ],
      "numa_node": 0
    },
    {
      "cpu": 0,
      "thread_siblings": [
        8,
        24
      ],
      "numa_node": 1
    },
    {
      "cpu": 6,
      "thread_siblings": [
        14,
        30
      ],
```

```
    "numa_node": 1
  },
  {
    "cpu": 3,
    "thread_siblings": [
      3,
      19
    ],
    "numa_node": 0
  },
  {
    "cpu": 2,
    "thread_siblings": [
      2,
      18
    ],
    "numa_node": 0
  }
],
"ram": [
  {
    "size_kb": 66980172,
    "numa_node": 0
  },
  {
    "size_kb": 67108864,
    "numa_node": 1
  }
],
"nics": [
  {
    "name": "ens3f1",
    "numa_node": 1
  },
  {
    "name": "ens3f0",
    "numa_node": 1
  },
  {
    "name": "ens2f0",
    "numa_node": 0
  },
  {
    "name": "ens2f1",
    "numa_node": 0
  },
  {
    "name": "ens1f1",
    "numa_node": 0
  },
  {
    "name": "ens1f0",
    "numa_node": 0
  },
  {
    "name": "eno4",
```

```
      "numa_node": 0
    },
    {
      "name": "eno1",
      "numa_node": 0
    },
    {
      "name": "eno3",
      "numa_node": 0
    },
    {
      "name": "eno2",
      "numa_node": 0
    }
  ]
}
```

CHAPTER 22. AUTOMATICALLY DISCOVERING BARE METAL NODES

You can use auto-discovery to register overcloud nodes and generate their metadata, without the need to create an **instackenv.json** file. This improvement can help to reduce the time it takes to collect information about a node. For example, if you use auto-discovery, you do not to collate the IPMI IP addresses and subsequently create the **instackenv.json**.

22.1. ENABLING AUTO-DISCOVERY

Enable and configure Bare Metal auto-discovery to automatically discover and import nodes that join your provisioning network when booting with PXE.

Procedure

1. Enable Bare Metal auto-discovery in the **undercloud.conf** file:

   ```
   enable_node_discovery = True
   discovery_default_driver = ipmi
   ```

   - **enable_node_discovery** - When enabled, any node that boots the introspection ramdisk using PXE is enrolled in the Bare Metal service (ironic) automatically.

   - **discovery_default_driver** - Sets the driver to use for discovered nodes. For example, **ipmi**.

2. Add your IPMI credentials to ironic:

   a. Add your IPMI credentials to a file named **ipmi-credentials.json**. Replace the **SampleUsername**, **RedactedSecurePassword**, and **bmc_address** values in this example to suit your environment:

   ```
   [
       {
           "description": "Set default IPMI credentials",
           "conditions": [
               {"op": "eq", "field": "data://auto_discovered", "value": true}
           ],
           "actions": [
               {"action": "set-attribute", "path": "driver_info/ipmi_username",
                "value": "SampleUsername"},
               {"action": "set-attribute", "path": "driver_info/ipmi_password",
                "value": "RedactedSecurePassword"},
               {"action": "set-attribute", "path": "driver_info/ipmi_address",
                "value": "{data[inventory][bmc_address]}"}
           ]
       }
   ]
   ```

3. Import the IPMI credentials file into ironic:

   ```
   $ openstack baremetal introspection rule import ipmi-credentials.json
   ```

22.2. TESTING AUTO-DISCOVERY

PXE boot a node that is connected to your provisioning network to test the Bare Metal auto-discovery feature.

Procedure

1. Power on the required nodes.

2. Run the **openstack baremetal node list** command. You should see the new nodes listed in an **enrolled** state:

```
$ openstack baremetal node list
+--------------------------------------+------+---------------+-------------+--------------------+-------------+
| UUID                                 | Name | Instance UUID | Power State | Provisioning State | Maintenance |
+--------------------------------------+------+---------------+-------------+--------------------+-------------+
| c6e63aec-e5ba-4d63-8d37-bd57628258e8 | None | None          | power off   | enroll             | False       |
| 0362b7b2-5b9c-4113-92e1-0b34a2535d9b | None | None          | power off   | enroll             | False       |
+--------------------------------------+------+---------------+-------------+--------------------+-------------+
```

3. Set the resource class for each node:

```
$ for NODE in `openstack baremetal node list -c UUID -f value` ; do openstack baremetal node set $NODE --resource-class baremetal ; done
```

4. Configure the kernel and ramdisk for each node:

```
$ for NODE in `openstack baremetal node list -c UUID -f value` ; do openstack baremetal node manage $NODE ; done
$ openstack overcloud node configure --all-manageable
```

5. Set all nodes to available:

```
$ for NODE in `openstack baremetal node list -c UUID -f value` ; do openstack baremetal node provide $NODE ; done
```

22.3. USING RULES TO DISCOVER DIFFERENT VENDOR HARDWARE

If you have a heterogeneous hardware environment, you can use introspection rules to assign credentials and remote management credentials. For example, you might want a separate discovery rule to handle your Dell nodes that use DRAC.

Procedure

1. Create a file named **dell-drac-rules.json** with the following contents:

```
[
    {
        "description": "Set default IPMI credentials",
```

```
        "conditions": [
            {"op": "eq", "field": "data://auto_discovered", "value": true},
            {"op": "ne", "field": "data://inventory.system_vendor.manufacturer",
             "value": "Dell Inc."}
        ],
        "actions": [
            {"action": "set-attribute", "path": "driver_info/ipmi_username",
             "value": "SampleUsername"},
            {"action": "set-attribute", "path": "driver_info/ipmi_password",
             "value": "RedactedSecurePassword"},
            {"action": "set-attribute", "path": "driver_info/ipmi_address",
             "value": "{data[inventory][bmc_address]}"}
        ]
    },
    {
        "description": "Set the vendor driver for Dell hardware",
        "conditions": [
            {"op": "eq", "field": "data://auto_discovered", "value": true},
            {"op": "eq", "field": "data://inventory.system_vendor.manufacturer",
             "value": "Dell Inc."}
        ],
        "actions": [
            {"action": "set-attribute", "path": "driver", "value": "idrac"},
            {"action": "set-attribute", "path": "driver_info/drac_username",
             "value": "SampleUsername"},
            {"action": "set-attribute", "path": "driver_info/drac_password",
             "value": "RedactedSecurePassword"},
            {"action": "set-attribute", "path": "driver_info/drac_address",
             "value": "{data[inventory][bmc_address]}"}
        ]
    }
]
```

- Replace the user name and password values in this example to suit your environment:

2. Import the rule into ironic:

```
$ openstack baremetal introspection rule import dell-drac-rules.json
```

CHAPTER 23. CONFIGURING AUTOMATIC PROFILE TAGGING

The introspection process performs a series of benchmark tests. The director saves the data from these tests. You can create a set of policies that use this data in various ways:

- The policies can identify underperforming or unstable nodes and isolate these nodes from use in the overcloud.

- The policies can define whether to tag nodes into specific profiles automatically.

23.1. POLICY FILE SYNTAX

Policy files use a JSON format that contains a set of rules. Each rule defines a description, a condition, and an action. A **description** is a plain text description of the rule, a **condition** defines an evaluation using a key-value pattern, and an **action** is the performance of the condition.

Description

A description is a plain text description of the rule.

Example:

> "description": "A new rule for my node tagging policy"

Conditions

A condition defines an evaluation using the following key-value pattern:

field

Defines the field to evaluate:

- **memory_mb** – The amount of memory for the node in MB.

- **cpus** – The total number of threads for the node CPU.

- **cpu_arch** – The architecture of the node CPU.

- **local_gb** – The total storage space of the node root disk.

op

Defines the operation to use for the evaluation. This includes the following attributes:

- **eq** - Equal to

- **ne** - Not equal to

- **lt** - Less than

- **gt** - Greater than

- **le** - Less than or equal to

- **ge** - Greater than or equal to

- **in-net** - Checks that an IP address is in a given network

- **matches** – Requires a full match against a given regular expression

- **contains** – Requires a value to contain a given regular expression

- **is-empty** – Checks that **field** is empty

invert

Boolean value to define whether to invert the result of the evaluation.

multiple

Defines the evaluation to use if multiple results exist. This parameter includes the following attributes:

- **any** – Requires any result to match

- **all** – Requires all results to match

- **first** – Requires the first result to match

value

Defines the value in the evaluation. If the field and operation result in the value, the condition return a true result. Otherwise, the condition returns a false result.

Example:

```
"conditions": [
  {
    "field": "local_gb",
    "op": "ge",
    "value": 1024
  }
],
```

Actions

If a condition is **true**, the policy performs an action. The action uses the **action** key and additional keys depending on the value of **action**:

- **fail** – Fails the introspection. Requires a **message** parameter for the failure message.

- **set-attribute** – Sets an attribute on an ironic node. Requires a **path** field, which is the path to an ironic attribute (for example, **/driver_info/ipmi_address**), and a **value** to set.

- **set-capability** – Sets a capability on an ironic node. Requires **name** and **value** fields, which are the name and the value for a new capability. This replaces the existing value for this capability. For example, use this to define node profiles.

- **extend-attribute** – The same as **set-attribute** but treats the existing value as a list and appends value to it. If the optional **unique** parameter is set to True, nothing is added if the given value is already in a list.

Example:

```
"actions": [
  {
    "action": "set-capability",
```

```
    "name": "profile",
    "value": "swift-storage"
  }
]
```

23.2. POLICY FILE EXAMPLE

The following is an example JSON file (**rules.json**) that contains introspection rules:

```
[
  {
    "description": "Fail introspection for unexpected nodes",
    "conditions": [
      {
        "op": "lt",
        "field": "memory_mb",
        "value": 4096
      }
    ],
    "actions": [
      {
        "action": "fail",
        "message": "Memory too low, expected at least 4 GiB"
      }
    ]
  },
  {
    "description": "Assign profile for object storage",
    "conditions": [
      {
        "op": "ge",
        "field": "local_gb",
        "value": 1024
      }
    ],
    "actions": [
      {
        "action": "set-capability",
        "name": "profile",
        "value": "swift-storage"
      }
    ]
  },
  {
    "description": "Assign possible profiles for compute and controller",
    "conditions": [
      {
        "op": "lt",
        "field": "local_gb",
        "value": 1024
      },
      {
        "op": "ge",
        "field": "local_gb",
        "value": 40
```

```
    }
  ],
  "actions": [
    {
      "action": "set-capability",
      "name": "compute_profile",
      "value": "1"
    },
    {
      "action": "set-capability",
      "name": "control_profile",
      "value": "1"
    },
    {
      "action": "set-capability",
      "name": "profile",
      "value": null
    }
  ]
  }
]
```

This example consists of three rules:

- Fail introspection if memory is lower than 4096 MiB. You can apply these types of rules if you want to exclude certain nodes from your cloud.

- Nodes with a hard drive size 1 TiB and bigger are assigned the swift-storage profile unconditionally.

- Nodes with a hard drive less than 1 TiB but more than 40 GiB can be either Compute or Controller nodes. You can assign two capabilities (**compute_profile** and **control_profile**) so that the **openstack overcloud profiles match** command can later make the final choice. For this process to succeed, you must remove the existing profile capability, otherwise the existing profile capability has priority.

The profile matching rules do not change any other nodes.

> **NOTE**
>
> Using introspection rules to assign the **profile** capability always overrides the existing value. However, **[PROFILE]_profile** capabilities are ignored for nodes that already have a profile capability.

23.3. IMPORTING POLICY FILES

To import policy files to director, complete the following steps.

Procedure

1. Import the policy file into director:

   ```
   $ openstack baremetal introspection rule import rules.json
   ```

2. Run the introspection process:

```
$ openstack overcloud node introspect --all-manageable
```

3. After introspection completes, check the nodes and their assigned profiles:

```
$ openstack overcloud profiles list
```

4. If you made a mistake in introspection rules, run the following command to delete all rules:

```
$ openstack baremetal introspection rule purge
```

CHAPTER 24. CREATING WHOLE-DISK IMAGES

The main overcloud image is a flat partition image that contains no partitioning information or bootloader. Director uses a separate kernel and ramdisk when it boots nodes and creates a basic partitioning layout when it writes the overcloud image to disk. However, you can create a whole- disk image, which includes a partitioning layout, bootloader, and hardened security.
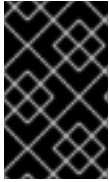
> **IMPORTANT**
>
> The following process uses the director image building feature. Red Hat only supports images that use the guidelines contained in this section. Custom images built outside of these specifications are not supported.

24.1. SECURITY HARDENING MEASURES

The whole disk image includes extra security hardening measures necessary for Red Hat OpenStack Platform deployments where security is an important feature.

Security recommendations for image creation

- The **/tmp** directory is mounted on a separate volume or partition and has the **rw**, **nosuid**, **nodev**, **noexec**, and **relatime** flags.

- The **/var**, **/var/log** and the **/var/log/audit** directories are mounted on separate volumes or partitions, with the **rw** and **relatime** flags.

- The **/home** directory is mounted on a separate partition or volume and has the **rw**, **nodev**, and **relatime** flags.

- Include the following changes to the **GRUB_CMDLINE_LINUX** setting:

  - To enable auditing, add the **audit=1** kernel boot flag.

  - To disable the kernel support for USB using boot loader configuration, add **nousb**.

  - To remove the insecure boot flags, remove **crashkernel=auto**.

- Blacklist insecure modules (**usb-storage**, **cramfs**, **freevxfs**, **jffs2**, **hfs**, **hfsplus**, **squashfs**, **udf**, **vfat**) and prevent these modules from loading.

- Remove any insecure packages like **telnet** from the image because they are installed by default.

24.2. WHOLE DISK IMAGE WORKFLOW

To build a whole disk image, complete the following workflow:

1. Download a base Red Hat Enterprise Linux 8.2 image.

2. Set the environment variables specific to registration.

3. Customize the image by modifying the partition schema and the size.

4. Create the image.

5. Upload the image to director.

24.3. DOWNLOADING THE BASE CLOUD IMAGE

Before you build a whole disk image, you must download an existing cloud image of Red Hat Enterprise Linux to use as a basis.

Procedure

1. Navigate to the Red Hat Enterprise Linux 8.2 download page. Red Hat OpenStack Platform 16.2 is supported on Red Hat Enterprise Linux 8.2.

   - https://access.redhat.com/downloads/content/479/ver=/rhel---8/8.4/x86_64/product-software

   > **NOTE**
   >
   > Enter your Customer Portal login details if a prompt appears.

2. Click **Download Now** next to **Red Hat Enterprise Linux 8.2 KVM Guest Image**

24.4. ENABLING CONSISTENT INTERFACE NAMING

Consistent network interface device naming is disabled in the KVM guest image by default. Use **virt-customize** to enable consistent naming.

Procedure

1. Move the KVM guest image to **/var/lib/libvirt/images**:

   ```
   $ sudo mv <kvm_guest_image> /var/lib/libvirt/images/
   ```

2. Start **libvirtd**:

   ```
   $ sudo systemctl start libvirtd
   ```

3. Enable consistent interface naming in the KVM guest image:

   ```
   $ sudo virt-customize -a /var/lib/libvirt/images/<kvm guest image> --edit
   /etc/default/grub:s/net.ifnames=0/net.ifnames=1/
   ```

4. Stop **libvirtd**:

   ```
   $ sudo systemctl stop libvirtd
   ```

24.5. DISK IMAGE ENVIRONMENT VARIABLES

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

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

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

DIB_LOCAL_IMAGE

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

REG_ACTIVATION_KEY

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

REG_AUTO_ATTACH

Defines whether to attach the most compatible subscription automatically.

REG_BASE_URL

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

REG_ENVIRONMENT

Registers to an environment within an organization.

REG_METHOD

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

REG_ORG

The organization where you want to register the images.

REG_POOL_ID

The pool ID of the product subscription information.

REG_PASSWORD

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

REG_RELEASE

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

REG_REPOS

A comma-separated string of repository names. Each repository in this string is enabled through **subscription-manager**.
Use the following repositories for a security hardened whole disk image:

- **rhel-8-for-x86_64-baseos-eus-rpms**

- **rhel-8-for-x86_64-appstream-eus-rpms**

- **rhel-8-for-x86_64-highavailability-eus-rpms**

- **ansible-2.9-for-rhel-8-x86_64-rpms**

- **fast-datapath-for-rhel-8-x86_64-rpms**

- **openstack-16.1-for-rhel-8-x86_64-rpms**

REG_SAT_URL

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

REG_SERVER_URL

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

REG_USER

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

Use the following set of example commands to export a set of environment variables and temporarily register a local QCOW2 image to the Red Hat Customer Portal:

```
$ export DIB_LOCAL_IMAGE=./rhel-8.2-x86_64-kvm.qcow2
$ export REG_METHOD=portal
$ export REG_USER=<your_name>
$ export REG_PASSWORD=<your_password>
$ export REG_RELEASE="8.2"
$ export REG_POOL_ID=<pool_id>
$ export REG_REPOS="rhel-8-for-x86_64-baseos-eus-rpms \
    rhel-8-for-x86_64-appstream-eus-rpms \
    rhel-8-for-x86_64-highavailability-eus-rpms \
    ansible-2.9-for-rhel-8-x86_64-rpms \
    fast-datapath-for-rhel-8-x86_64-rpms \
    openstack-16.1-for-rhel-8-x86_64-rpms"
```

24.6. CUSTOMIZING THE DISK LAYOUT

The default security hardened image size is 20G and uses predefined partitioning sizes. However, you must modify the partitioning layout to accommodate overcloud container images. Complete the steps in the following sections to increase the image size to 40G. You can modify the partitioning layout and disk size to further suit your needs.

To modify the partitioning layout and disk size, perform the following steps:

- Modify the partitioning schema using the **DIB_BLOCK_DEVICE_CONFIG** environment variable.

- Modify the global size of the image by updating the **DIB_IMAGE_SIZE** environment variable.

24.7. MODIFYING THE PARTITIONING SCHEMA

You can modify the partitioning schema to alter the partitioning size, create new partitions, or remove existing partitions. Use the following environment variable to define a new partitioning schema:

```
$ export DIB_BLOCK_DEVICE_CONFIG='<yaml_schema_with_partitions>'
```

BIOS example

The following YAML structure represents the modified logical volume partitioning layout to accommodate enough space to pull overcloud container images:

```
export DIB_BLOCK_DEVICE_CONFIG='''
- local_loop:
    name: image0
- partitioning:
    base: image0
    label: mbr
    partitions:
      - name: root
        flags: [ boot,primary ]
        size: 40G
- lvm:
    name: lvm
    base: [ root ]
    pvs:
      - name: pv
        base: root
        options: [ "--force" ]
    vgs:
      - name: vg
        base: [ "pv" ]
        options: [ "--force" ]
    lvs:
      - name: lv_root
        base: vg
        extents: 23%VG
      - name: lv_tmp
        base: vg
        extents: 4%VG
      - name: lv_var
        base: vg
        extents: 45%VG
      - name: lv_log
        base: vg
        extents: 23%VG
      - name: lv_audit
        base: vg
        extents: 4%VG
      - name: lv_home
        base: vg
        extents: 1%VG
- mkfs:
    name: fs_root
    base: lv_root
    type: xfs
    label: "img-rootfs"
    mount:
        mount_point: /
        fstab:
            options: "rw,relatime"
            fsck-passno: 1
- mkfs:
    name: fs_tmp
    base: lv_tmp
    type: xfs
    mount:
        mount_point: /tmp
```

```
        fstab:
            options: "rw,nosuid,nodev,noexec,relatime"
            fsck-passno: 2
  - mkfs:
      name: fs_var
      base: lv_var
      type: xfs
      mount:
          mount_point: /var
          fstab:
              options: "rw,relatime"
              fsck-passno: 2
  - mkfs:
      name: fs_log
      base: lv_log
      type: xfs
      mount:
          mount_point: /var/log
          fstab:
              options: "rw,relatime"
              fsck-passno: 3
  - mkfs:
      name: fs_audit
      base: lv_audit
      type: xfs
      mount:
          mount_point: /var/log/audit
          fstab:
              options: "rw,relatime"
              fsck-passno: 4
  - mkfs:
      name: fs_home
      base: lv_home
      type: xfs
      mount:
          mount_point: /home
          fstab:
              options: "rw,nodev,relatime"
              fsck-passno: 2
  '''
```

Use this sample YAML content as a basis for the partition schema of your image. Modify the partition sizes and layout to suit your needs.

> **NOTE**
>
> You must define the correct partition sizes for the image because you cannot resize them after the deployment.

UEFI example

The following YAML structure represents the modified logical volume partitioning layout to accommodate enough space to pull overcloud container images:

```
export DIB_BLOCK_DEVICE_CONFIG='''
```

```
        - local_loop:
            name: image0
        - partitioning:
            base: image0
            label: gpt
            partitions:
              - name: ESP
                type: 'EF00'
                size: 200MiB
              - name: BSP
                type: 'EF02'
                size: 1MiB
              - name: ROOT
                type: '8300'
                size: 100%
        - mkfs:
            name: fs_esp
            base: ESP
            type: vfat
            mount:
              mount_point: /boot/efi
              fstab:
                options: "defaults"
                fsck-passno: 1
        - mkfs:
            name: fs_root
            label: "img-rootfs"
            base: ROOT
            type: xfs
            mount:
              mount_point: /
              fstab:
                options: "defaults"
                fsck-passno: 1
        '''
```

Use this sample YAML content as a basis for the partition schema of your image. Modify the partition sizes and layout to suit your environment.

NOTE

You must define the correct partition sizes for the image before deployment because you cannot resize them after the deployment.

24.8. MODIFYING THE IMAGE SIZE

The global sum of the modified partitioning schema might exceed the default disk size (20G). In this situation, you might need to modify the image size. To modify the image size, edit the configuration files that create the image.

Procedure

1. Create a copy of the **/usr/share/openstack-tripleo-common/image-yaml/overcloud-hardened-images-python3.yaml**:

```
# cp /usr/share/openstack-tripleo-common/image-yaml/overcloud-hardened-images-
python3.yaml \
/home/stack/overcloud-hardened-images-python3-custom.yaml
```
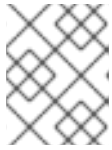
> **NOTE**
>
> For UEFI whole disk images, use **/usr/share/openstack-tripleo-common/image-yaml/overcloud-hardened-images-uefi-python3.yaml**.

2. Edit the **DIB_IMAGE_SIZE** in the configuration file and adjust the values as necessary:

```
...

environment:
  DIB_PYTHON_VERSION: '3'
  DIB_MODPROBE_BLACKLIST: 'usb-storage cramfs freevxfs jffs2 hfs hfsplus squashfs udf
vfat bluetooth'
  DIB_BOOTLOADER_DEFAULT_CMDLINE: 'nofb nomodeset vga=normal console=tty0
console=ttyS0,115200 audit=1 nousb'
  DIB_IMAGE_SIZE: '40' ❶
  COMPRESS_IMAGE: '1'
```
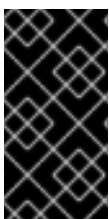
❶ Adjust this value to the new total disk size.

3. Optional. To configure a proxy, you must also include the **http_proxy** and **https:_proxy** environment variables:

```
environment:
  http_proxy: <proxy_server>
  https_proxy: <proxy_server>
```

- Replace **<proxy_server>** with the address of your proxy.

4. Save the file.

> **IMPORTANT**
>
> When you deploy the overcloud, the director creates a RAW version of the overcloud image. This means your undercloud must have enough free space to accommodate the RAW image. For example, if you set the security hardened image size to 40G, you must have 40G of space available on the undercloud hard disk.
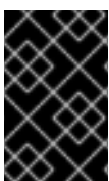
> **IMPORTANT**
>
> When director writes the image to the physical disk, it creates a 64MB configuration drive primary partition at the end of the disk. When you create your whole disk image, ensure that the size of the physical disk accommodates this extra partition.

24.9. BUILDING THE WHOLE DISK IMAGE

After you set the environment variables and customize the image, create the image using the **openstack overcloud image build** command.

Procedure

1. Run the **openstack overcloud image build** command with all necessary configuration files.

```
# openstack overcloud image build \
--image-name overcloud-hardened-full \ 1
--config-file /home/stack/overcloud-hardened-images-python3-custom.yaml \ 2
--config-file /usr/share/openstack-tripleo-common/image-yaml/overcloud-hardened-images-
rhel8.yaml 3
```

1 For UEFI whole disk images, use **overcloud-hardened-uefi-full**.

2 The **overcloud-hardened-images-python3-custom.yaml** file is the custom configuration file that contains the new disk size. If you are not using a different custom disk size, use the original **/usr/share/openstack-tripleo-common/image-yaml/overcloud-hardened-images-python3.yaml** file instead. For standard UEFI whole disk images, use **overcloud-hardened-images-uefi-python3.yaml**.

3 For UEFI whole disk images, use **overcloud-hardened-images-uefi-rhel8.yaml**.

This command creates an image called **overcloud-hardened-full.qcow2**, which contains all the necessary security features.

24.10. UPLOADING THE WHOLE DISK IMAGE

Upload the image to the OpenStack Image (glance) service and start using it from the Red Hat OpenStack Platform director. To upload a security hardened image, complete the following steps:

1. Rename the newly generated image and move the image to your **images** directory:

```
# mv overcloud-hardened-full.qcow2 ~/images/overcloud-full.qcow2
```

2. Remove all the old overcloud images:

```
# openstack image delete overcloud-full
# openstack image delete overcloud-full-initrd
# openstack image delete overcloud-full-vmlinuz
```

3. Upload the new overcloud image:

```
# openstack overcloud image upload --image-path /home/stack/images --whole-disk
```

If you want to replace an existing image with the security hardened image, use the **--update-existing** flag. This flag overwrites the original **overcloud-full** image with a new security hardened image.

CHAPTER 25. CONFIGURING DIRECT DEPLOY

When provisioning nodes, director mounts the overcloud base operating system image on an iSCSI mount and then copies the image to disk on each node. Direct deploy is an alternative method that writes disk images from a HTTP location directly to disk on bare metal nodes.

25.1. CONFIGURING THE DIRECT DEPLOY INTERFACE ON THE UNDERCLOUD

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

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

Procedure

1. Create or modify a custom environment file **/home/stack/undercloud_custom_env.yaml** and specify the **IronicDefaultDeployInterface**.

   ```
   parameter_defaults:
     IronicDefaultDeployInterface: direct
   ```

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

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

3. Include the custom environment file in the **DEFAULT** section of the **undercloud.conf** file.

   ```
   custom_env_files = /home/stack/undercloud_custom_env.yaml
   ```

4. Perform the undercloud installation:

   ```
   $ openstack undercloud install
   ```

CHAPTER 26. CREATING VIRTUALIZED CONTROL PLANES

A virtualized control plane is a control plane located on virtual machines (VMs) rather than on bare metal. Use a virtualized control plane reduce the number of bare metal machines that you require for the control plane.

This chapter explains how to virtualize your Red Hat OpenStack Platform (RHOSP) control plane for the overcloud using RHOSP and Red Hat Virtualization.

26.1. VIRTUALIZED CONTROL PLANE ARCHITECTURE

Use director to provision an overcloud using Controller nodes that are deployed in a Red Hat Virtualization cluster. You can then deploy these virtualized controllers as the virtualized control plane nodes.

> **NOTE**
>
> Virtualized Controller nodes are supported only on Red Hat Virtualization.
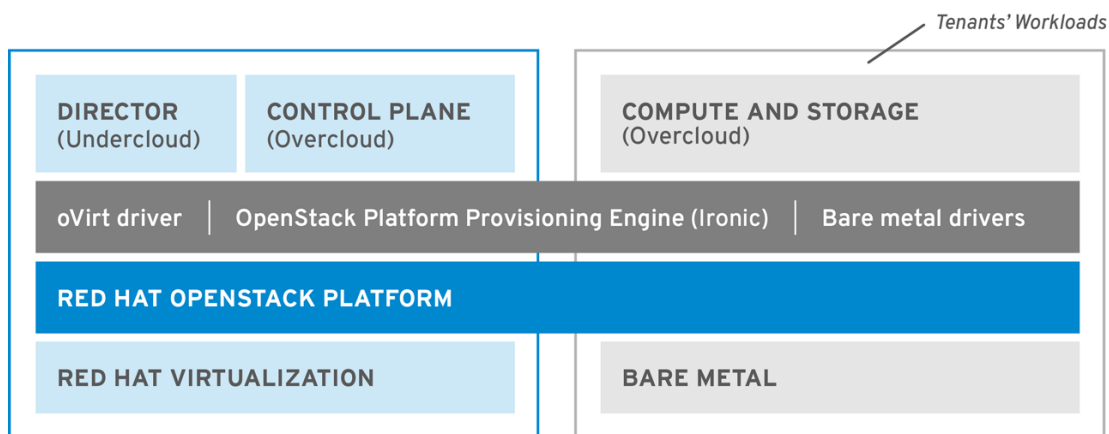
The following architecture diagram illustrates how to deploy a virtualized control plane. Distribute the overcloud with the Controller nodes running on VMs on Red Hat Virtualization and run the Compute and Storage nodes on bare metal.

> **NOTE**
>
> Run the OpenStack virtualized undercloud on Red Hat Virtualization.

Virtualized control plane architecture

The OpenStack Bare Metal Provisioning service (ironic) includes a driver for Red Hat Virtualization VMs, **staging-ovirt**. You can use this driver to manage virtual nodes within a Red Hat Virtualization environment. You can also use it to deploy overcloud controllers as virtual machines within a Red Hat Virtualization environment.

Benefits and limitations of virtualizing your RHOSP overcloud control plane

Although there are a number of benefits to virtualizing your RHOSP overcloud control plane, this is not an option in every configuration.

Benefits

Virtualizing the overcloud control plane has a number of benefits that prevent downtime and improve performance.

- You can allocate resources to the virtualized controllers dynamically, using hot add and hot remove to scale CPU and memory as required. This prevents downtime and facilitates increased capacity as the platform grows.

- You can deploy additional infrastructure VMs on the same Red Hat Virtualization cluster. This minimizes the server footprint in the data center and maximizes the efficiency of the physical nodes.

- You can use composable roles to define more complex RHOSP control planes and allocate resources to specific components of the control plane.

- You can maintain systems without service interruption with the VM live migration feature.

- You can integrate third-party or custom tools that Red Hat Virtualization supports.

Limitations

Virtualized control planes limit the types of configurations that you can use.

- Virtualized Ceph Storage nodes and Compute nodes are not supported.

- Block Storage (cinder) image-to-volume is not supported for back ends that use Fiber Channel. Red Hat Virtualization does not support N_Port ID Virtualization (NPIV). Therefore, Block Storage (cinder) drivers that need to map LUNs from a storage back end to the controllers, where **cinder-volume** runs by default, do not work. You must create a dedicated role for **cinder-volume** and use the role to create physical nodes instead of including it on the virtualized controllers. For more information, see Composable Services and Custom Roles.

26.2. PROVISIONING VIRTUALIZED CONTROLLERS USING THE RED HAT VIRTUALIZATION DRIVER

Complete the following steps to provision a virtualized RHOSP control plane for the overcloud using RHOSP and Red Hat Virtualization.

Prerequisites

- You must have a 64-bit x86 processor with support for the Intel 64 or AMD64 CPU extensions.

- You must have the following software already installed and configured:

  - Red Hat Virtualization. For more information, see Red Hat Virtualization Documentation Suite.

  - Red Hat OpenStack Platform (RHOSP). For more information, see Director Installation and Usage.

- You must have the virtualized Controller nodes prepared in advance. These requirements are the same as for bare metal Controller nodes. For more information, see Controller Node Requirements.

- You must have the bare metal nodes being used as overcloud Compute nodes, and the storage nodes, prepared in advance. For hardware specifications, see the Compute Node Requirements

and Ceph Storage Node Requirements . To deploy overcloud Compute nodes on POWER (ppc64le) hardware, see Red Hat OpenStack Platform for POWER .

- You must have the logical networks created, and your cluster of host networks ready to use network isolation with multiple networks. For more information, see Logical Networks.

- You must have the internal BIOS clock of each node set to UTC to prevent issues with future-dated file timestamps when hwclock synchronizes the BIOS clock before applying the timezone offset.

TIP

To avoid performance bottlenecks, use composable roles and keep the data plane services on the bare metal Controller nodes.

Procedure

1. To enable the **staging-ovirt** driver in director, add the driver to the **enabled_hardware_types** parameter in the **undercloud.conf** configuration file:

   enabled_hardware_types = ipmi,redfish,ilo,idrac,staging-ovirt

2. Verify that the undercloud contains the **staging-ovirt** driver:

   (undercloud) [stack@undercloud ~]$ openstack baremetal driver list

   If you have configured the undercloud correctly, this command returns the following result:

   ```
   +--------------------+----------------------+
   | Supported driver(s) | Active host(s)       |
   +--------------------+----------------------+
   | idrac              | localhost.localdomain |
   | ilo                | localhost.localdomain |
   | ipmi               | localhost.localdomain |
   | pxe_drac           | localhost.localdomain |
   | pxe_ilo            | localhost.localdomain |
   | pxe_ipmitool       | localhost.localdomain |
   | redfish            | localhost.localdomain |
   | staging-ovirt      | localhost.localdomain |
   ```

3. Update the overcloud node definition template, for example, **nodes.json**, to register the VMs hosted on Red Hat Virtualization with director. For more information, see Registering Nodes for the Overcloud. Use the following key:value pairs to define aspects of the VMs that you want to deploy with your overcloud:

   Table 26.1. Configuring the VMs for the overcloud

Key	Set to this value
pm_type	OpenStack Bare Metal Provisioning (ironic) service driver for oVirt/RHV VMs, **staging-ovirt**.

Key	Set to this value
pm_user	Red Hat Virtualization Manager username.
pm_password	Red Hat Virtualization Manager password.
pm_addr	Hostname or IP of the Red Hat Virtualization Manager server.
pm_vm_name	Name of the virtual machine in Red Hat Virtualization Manager where the controller is created.

For example:

```
{
    "nodes": [
      {
          "name":"osp13-controller-0",
          "pm_type":"staging-ovirt",
          "mac":[
             "00:1a:4a:16:01:56"
          ],
          "cpu":"2",
          "memory":"4096",
          "disk":"40",
          "arch":"x86_64",
          "pm_user":"admin@internal",
          "pm_password":"password",
          "pm_addr":"rhvm.example.com",
          "pm_vm_name":"{osp_curr_ver}-controller-0",
          "capabilities": "profile:control,boot_option:local"
      },
      ...
  }
```

Configure one Controller on each Red Hat Virtualization Host

4. Configure an affinity group in Red Hat Virtualization with "soft negative affinity" to ensure high availability is implemented for your controller VMs. For more information, see Affinity Groups.

5. Open the Red Hat Virtualization Manager interface, and use it to map each VLAN to a separate logical vNIC in the controller VMs. For more information, see Logical Networks.

6. Set **no_filter** in the vNIC of the director and controller VMs, and restart the VMs, to disable the MAC spoofing filter on the networks attached to the controller VMs. For more information, see Virtual Network Interface Cards .

7. Deploy the overcloud to include the new virtualized controller nodes in your environment:

```
(undercloud) [stack@undercloud ~]$ openstack overcloud deploy --templates
```

CHAPTER 27. PERFORMING ADVANCED CONTAINER IMAGE MANAGEMENT

The default container image configuration suits most environments. In some situations, your container image configuration might require some customization, such as version pinning.

27.1. PINNING CONTAINER IMAGES FOR THE UNDERCLOUD

In certain circumstances, you might require a set of specific container image versions for your undercloud. In this situation, you must pin the images to a specific version. To pin your images, you must generate and modify a container configuration file, and then combine the undercloud roles data with the container configuration file to generate an environment file that contains a mapping of services to container images. Then include this environment file in the **custom_env_files** parameter in the **undercloud.conf** file.

Procedure

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

2. Run the **openstack tripleo container image prepare default** command with the **--output-env-file** option to generate a file that contains the default image configuration:

   ```
   $ sudo openstack tripleo container image prepare default \
   --output-env-file undercloud-container-image-prepare.yaml
   ```

3. Modify the **undercloud-container-image-prepare.yaml** file according to the requirements of your environment.

   a. Remove the **tag:** parameter so that director can use the **tag_from_label:** parameter. Director uses this parameter to identify the latest version of each container image, pull each image, and tag each image on the container registry in director.

   b. Remove the Ceph labels for the undercloud.

   c. Ensure that the **neutron_driver:** parameter is empty. Do not set this parameter to **OVN** because OVN is not supported on the undercloud.

   d. Include your container image registry credentials:

   ```
   ContainerImageRegistryCredentials:
     registry.redhat.io
       myser: 'p@55w0rd!'
   ```

   > **NOTE**
   >
   > You cannot push container images to the undercloud registry on new underclouds because the **image-serve** registry is not installed yet. You must set the **push_destination** value to **false**, or use a custom value, to pull images directly from source. For more information, see Container image preparation parameters.

4. Generate a new container image configuration file that uses the undercloud roles file combined with your custom **undercloud-container-image-prepare.yaml** file:

```
$ sudo openstack tripleo container image prepare \
-r /usr/share/openstack-tripleo-heat-templates/roles_data_undercloud.yaml \
-e undercloud-container-image-prepare.yaml \
--output-env-file undercloud-container-images.yaml
```

The **undercloud-container-images.yaml** file is an environment file that contains a mapping of service parameters to container images. For example, OpenStack Identity (keystone) uses the **ContainerKeystoneImage** parameter to define its container image:

```
ContainerKeystoneImage: undercloud.ctlplane.localdomain:8787/rhosp-rhel8/openstack-keystone:16.1.4-5
```

Note that the container image tag matches the **{version}-{release}** format.

5. Include the **undercloud-container-images.yaml** file in the **custom_env_files** parameter in the **undercloud.conf** file. When you run the undercloud installation, the undercloud services use the pinned container image mapping from this file.

27.2. PINNING CONTAINER IMAGES FOR THE OVERCLOUD

In certain circumstances, you might require a set of specific container image versions for your overcloud. In this situation, you must pin the images to a specific version. To pin your images, you must create the **containers-prepare-parameter.yaml** file, use this file to pull your container images to the undercloud registry, and generate an environment file that contains a pinned image list.

For example, your **containers-prepare-parameter.yaml** file might contain the following content:

```
parameter_defaults:
  ContainerImagePrepare:
  - push_destination: true
    set:
      name_prefix: openstack-
      name_suffix: ''
      namespace: registry.redhat.io/rhosp-rhel8
      neutron_driver: ovn
    tag_from_label: '{version}-{release}'

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

The **ContainerImagePrepare** parameter contains a single rule **set**. This rule **set** must not include the **tag** parameter and must rely on the **tag_from_label** parameter to identify the latest version and release of each container image. Director uses this rule **set** to identify the latest version of each container image, pull each image, and tag each image on the container registry in director.

Procedure

1. Run the **openstack tripleo container image prepare** command, which pulls all images from the source defined in the **containers-prepare-parameter.yaml** file. Include the **--output-env-file** to specify the output file that will contain the list of pinned container images:

```
$ sudo openstack tripleo container image prepare -e /home/stack/templates/containers-prepare-parameter.yaml --output-env-file overcloud-images.yaml
```

The **overcloud-images.yaml** file is an environment file that contains a mapping of service parameters to container images. For example, OpenStack Identity (keystone) uses the **ContainerKeystoneImage** parameter to define its container image:

```
ContainerKeystoneImage: undercloud.ctlplane.localdomain:8787/rhosp-rhel8/openstack-keystone:16.2.4-5
```

Note that the container image tag matches the **{version}-{release}** format.

2. Include the **containers-prepare-parameter.yaml** and **overcloud-images.yaml** files in that specific order with your environment file collection when you run the **openstack overcloud deploy** command:

```
$ openstack overcloud deploy --templates \
    ...
    -e /home/stack/containers-prepare-parameter.yaml \
    -e /home/stack/overcloud-images.yaml \
    ...
```

The overcloud services use the pinned images listed in the **overcloud-images.yaml** file.

CHAPTER 28. TROUBLESHOOTING DIRECTOR ERRORS

Errors can occur at certain stages of the director processes. This section contains some information about diagnosing common problems.

28.1. 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]
      ```

28.2. TROUBLESHOOTING HARDWARE INTROSPECTION

You must run the introspection process to completion. However, **ironic-inspector** times out after a default one hour period if the inspection ramdisk does not respond. Sometimes this indicates a bug in the inspection ramdisk but usually this time-out occurs due to an environment misconfiguration, particularly BIOS boot settings.

To diagnose and resolve common environment misconfiguration issues, complete the following steps:

Procedure

1. Source the **stackrc** file:

   ```
   $ source ~/stackrc
   ```

2. Director uses OpenStack Object Storage (swift) to save the hardware data that it obtains during the introspection process. If this service is not running, the introspection can fail. Check all services related to OpenStack Object Storage to ensure that the service is running:

   ```
   (undercloud) $ sudo systemctl list-units tripleo_swift*
   ```

3. 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]
   ```

4. Configure temporary access to the introspection ramdisk. You can provide either a temporary password or an SSH key to access the node during introspection debugging. Complete the following procedure to configure ramdisk access:

   a. Run the **openssl passwd -1** command with a temporary password to generate an MD5 hash:

      ```
      (undercloud) $ openssl passwd -1 mytestpassword
      $1$enjRSyIw$/fYUpJwr6abFy/d.koRgQ/
      ```

   b. Edit the **/var/lib/ironic/httpboot/inspector.ipxe** file, find the line starting with **kernel**, and append the **rootpwd** parameter and the MD5 hash:

      ```
      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
      rootpwd="$1$enjRSyIw$/fYUpJwr6abFy/d.koRgQ/" selinux=0
      ```

      Alternatively, append your public SSH key to the **sshkey** parameter.
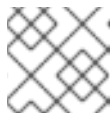
      > **NOTE**
      >
      > Include quotation marks for both the **rootpwd** and **sshkey** parameters.

5. 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.

6. Identify the IP address of the node from the **dnsmasq** logs:

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

7. 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.

8. 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.

28.3. TROUBLESHOOTING WORKFLOWS AND EXECUTIONS

The OpenStack Workflow (mistral) service groups multiple OpenStack tasks into workflows. Red Hat OpenStack Platform uses a set of these workflows to perform common functions across the director, including bare metal node control, validations, plan management, and overcloud deployment.

For example, when you run the **openstack overcloud deploy** command, the OpenStack Workflow service executes two workflows. The first workflow uploads the deployment plan:

> Removing the current plan files
> Uploading new plan files
> Started Mistral Workflow. Execution ID: aef1e8c6-a862-42de-8bce-073744ed5e6b
> Plan updated

The second workflow starts the overcloud deployment:

> Deploying templates in the directory /tmp/tripleoclient-LhRlHX/tripleo-heat-templates
> Started Mistral Workflow. Execution ID: 97b64abe-d8fc-414a-837a-1380631c764d
> 2016-11-28 06:29:26Z [overcloud]: CREATE_IN_PROGRESS Stack CREATE started
> 2016-11-28 06:29:26Z [overcloud.Networks]: CREATE_IN_PROGRESS state changed
> 2016-11-28 06:29:26Z [overcloud.HeatAuthEncryptionKey]: CREATE_IN_PROGRESS state changed
> 2016-11-28 06:29:26Z [overcloud.ServiceNetMap]: CREATE_IN_PROGRESS state changed
> ...

The OpenStack Workflow service uses the following objects to track the workflow:

Actions

A particular instruction that OpenStack performs when an associated task runs. Examples include running shell scripts or performing HTTP requests. Some OpenStack components have in–built actions that OpenStack Workflow uses.

Tasks

Defines the action to run and the result of running the action. These tasks usually have actions or other workflows associated with them. When a task completes, the workflow directs to another task, usually depending on whether the task succeeded or failed.

Workflows

A set of tasks grouped together and executed in a specific order.

Executions

Defines a particular action, task, or workflow running.

OpenStack Workflow also provides robust logging of executions, which helps to identify issues with certain command failures. For example, if a workflow execution fails, you can identify the point of failure.

Procedure

1. Source the **stackrc** file:

   $ source ~/stackrc

2. List the workflow executions that have the failed state **ERROR**:

   (undercloud) $ openstack workflow execution list | grep "ERROR"

3. Get the UUID of the failed workflow execution (for example, **dffa96b0-f679-4cd2-a490-4769a3825262**) and view the execution and output:

   (undercloud) $ openstack workflow execution show dffa96b0-f679-4cd2-a490-4769a3825262
   (undercloud) $ openstack workflow execution output show dffa96b0-f679-4cd2-a490-4769a3825262

4. These commands return information about the failed task in the execution. The **openstack workflow execution show** command also displays the workflow that was used for the execution (for example, **tripleo.plan_management.v1.publish_ui_logs_to_swift**). You can view the full workflow definition with the following command:

   (undercloud) $ openstack workflow definition show tripleo.plan_management.v1.publish_ui_logs_to_swift

   This is useful for identifying where in the workflow a particular task occurs.

5. View action executions and their results using a similar command syntax:

   (undercloud) $ openstack action execution list
   (undercloud) $ openstack action execution show 8a68eba3-0fec-4b2a-adc9-5561b007e886
   (undercloud) $ openstack action execution output show 8a68eba3-0fec-4b2a-adc9-5561b007e886

   This is useful for identifying a specific action that causes issues.

28.4. 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. Run the deployment failures command:

   ```
   $ openstack overcloud failures
   ```

3. Run the following command to display the details of the failure:

   ```
   (undercloud) $ openstack stack failures list <OVERCLOUD_NAME> --long
   ```

   - Replace **<OVERCLOUD_NAME>** with the name of your overcloud.

4. Run the following command to identify the stacks that failed:

   ```
   (undercloud) $ openstack stack list --nested --property status=FAILED
   ```

28.5. 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.	The 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 the node details including the profile and flavor mapping. Check that the node hardware details are within the requirements for the flavor.
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](#)

28.6. 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.

28.7. TROUBLESHOOTING "NO VALID HOST FOUND" ERRORS

Sometimes the **/var/log/nova/nova-conductor.log** contains the following error:

> NoValidHost: No valid host was found. There are not enough hosts available.

This error occurs when the Compute Scheduler cannot find a bare metal node that is suitable for booting the new instance. This usually means that there is a mismatch between resources that the Compute service expects to find and resources that the Bare Metal service advertised to Compute. To check that there is a mismatch error, complete the following steps:

Procedure

1. Source the **stackrc** file:

   ```
   $ source ~/stackrc
   ```

2. Check that the introspection succeeded on the node. If the introspection fails, check that each node contains the required ironic node properties:

   ```
   (undercloud) $ openstack baremetal node show [NODE UUID]
   ```

   Check that the **properties** JSON field has valid values for keys **cpus**, **cpu_arch**, **memory_mb** and **local_gb**.

3. Ensure that the Compute flavor that is mapped to the node does not exceed the node properties for the required number of nodes:

   ```
   (undercloud) $ openstack flavor show [FLAVOR NAME]
   ```

4. Run the **openstack baremetal node list** command to ensure that there are sufficient nodes in the available state. Nodes in **manageable** state usually signify a failed introspection.

5. Run the **openstack baremetal node list** command and ensure that the nodes are not in maintenance mode. If a node changes to maintenance mode automatically, the likely cause is an issue with incorrect power management credentials. Check the power management credentials and then remove maintenance mode:

   (undercloud) $ openstack baremetal node maintenance unset [NODE UUID]

6. If you are using automatic profile tagging, check that you have enough nodes that correspond to each flavor and profile. Run the **openstack baremetal node show** command on a node and check the **capabilities** key in the **properties** field. For example, a node tagged for the Compute role contains the **profile:compute** value.

7. You must wait for node information to propagate from Bare Metal to Compute after introspection. However, if you performed some steps manually, there might be a short period of time when nodes are not available to the Compute service (nova). Use the following command to check the total resources in your system:

   (undercloud) $ openstack hypervisor stats show

28.8. 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 **/var/lib/mistral** directory on the **undercloud**:

   $ sudo setfacl -R -m u:stack:rwx /var/lib/mistral

   This command retains **mistral** user access to the directory.

2. Change to the working directory for the **config-download** files. This is usually **/var/lib/mistral/overcloud/**.

   $ cd /var/lib/mistral/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 ocurred.

28.9. TROUBLESHOOTING CONTAINER CONFIGURATION

Red Hat OpenStack Platform director uses **paunch** to launch containers, **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) $ openstack server list

3. Log in to the node:

   (undercloud) $ ssh heat-admin@192.168.24.60

4. Change to the root user:

   $ sudo -i

Identifying failed containers

1. View all containers:

   $ 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>
```
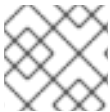
> **NOTE**
>
> The **-ti** options run the command through an interactive pseudoterminal.

- 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:

   ```
   $ podman mount keystone
   ```

   This provides a mounted location to view the filesystem contents:

   ```
   /var/lib/containers/storage/overlay/78946a109085aeb8b3a350fc20bd8049a08918d74f573396d7358270e711c610/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.

28.10. 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 heat-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.

28.11. 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"

28.12. LOG LOCATIONS

Use the following logs to gather information about the undercloud and overcloud when you troubleshoot issues.

Table 28.1. 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	**/var/lib/mistral/overcloud/ansible.log**

Table 28.2. 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 28.3. 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 29. TIPS FOR UNDERCLOUD AND OVERCLOUD SERVICES

This section provides advice on tuning and managing specific OpenStack services on the undercloud.

29.1. 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
```

29.2. RUNNING SWIFT-RING-BUILDER IN A CONTAINER

To manage your Object Storage (swift) rings, use the **swift-ring-builder** commands inside the server containers:

- **swift_object_server**

- **swift_container_server**

- **swift_account_server**

For example, to view information about your swift object rings, run the following command:

```
$ sudo podman exec -ti -u swift swift_object_server swift-ring-builder /etc/swift/object.builder
```

You can run this command on both the undercloud and overcloud nodes.

29.3. CHANGING THE SSL/TLS CIPHER RULES FOR HAPROXY

If you enabled SSL/TLS in the undercloud (see Section 4.2, "Director 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 30. 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.

30.1. 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.

30.2. 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.

30.3. 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.

30.4. 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**.
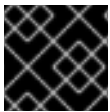
pm_user; pm_password

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.

30.5. 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.

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.

30.6. RED HAT VIRTUALIZATION

This driver provides control over virtual machines in Red Hat Virtualization (RHV) through its RESTful API.

pm_type

Set this option to **staging-ovirt**.

pm_user; pm_password

The username and password for your RHV environment. The username also includes the authentication provider. For example: **admin@internal**.

pm_addr

The IP address of the RHV REST API.

pm_vm_name

The name of the virtual machine to control.

mac

A list of MAC addresses for the network interfaces on the node. Use only the MAC address for the Provisioning NIC of each system.

- To enable this driver, add **staging-ovirt** to the **enabled_hardware_types** option in your **undercloud.conf** and rerun the **openstack undercloud install** command.

30.7. 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.