Red Hat OpenStack Platform 17.0

Director Installation and Usage

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

Last Updated: 2024-02-12

Red Hat OpenStack Platform 17.0 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 17 in an enterprise environment using the Red Hat OpenStack Platform director. This includes installing the director, planning your environment, and creating an OpenStack environment with the director.

Table of Contents

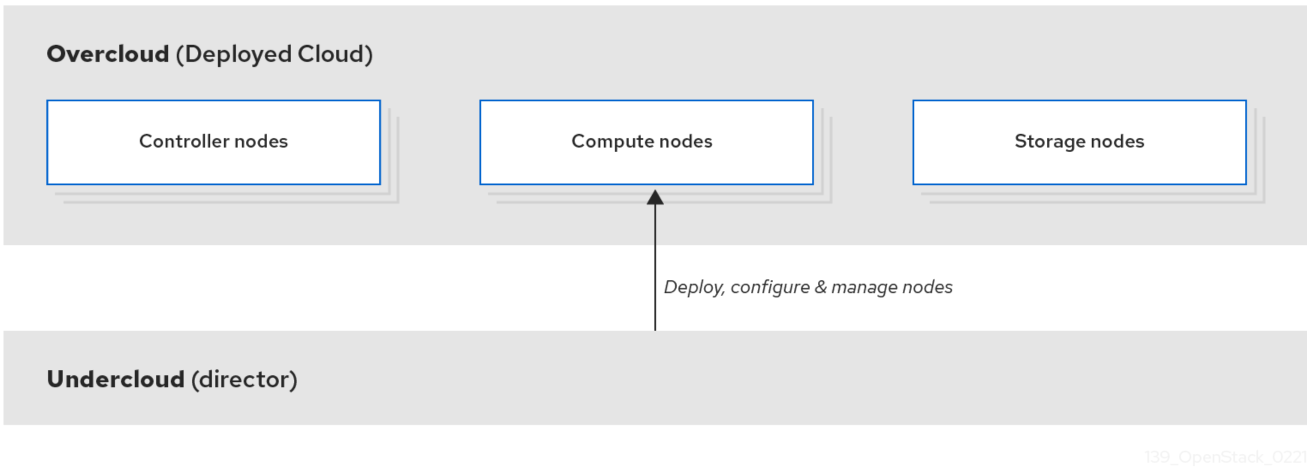
MAKING OPEN SOURCE MORE INCLUSIVE

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

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.



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) - Manages bare metal nodes.
- OpenStack Networking (neutron) and Open vSwitch - Control networking for bare metal nodes.
- OpenStack Orchestration (Ephemeral Heat) - Provides the orchestration of nodes after director writes the overcloud image to disk.

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
- 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 17.0 supports installation on the Red Hat Enterprise Linux 9.0 operating system. Red Hat OpenStack Platform previously used Docker to manage containerization. OpenStack Platform 17.0 uses these 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. 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 .

1.6. DEFAULT FILE LOCATIONS

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

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

1.6.1. Description of the contents of the undercloud directory

You can find the following files and directories in the **~/tripleo-deploy/undercloud** directory. They are a subset of what is available in the **~/overcloud-deploy** directory:

```
heat_launcher
install-undercloud.log
tripleo-ansible-inventory.yaml
tripleo-config-generated-env-files
tripleo-undercloud-outputs.yaml
tripleo-undercloud-passwords.yaml
undercloud-install-20220823210316.tar.bzip2
```

1.6.2. Description of the contents of the overcloud directory

You can find the following files and directories in the ~/**overcloud-deploy**/**overcloud** directory, where **overcloud** is the name of the stack:

```
cli-config-download
cli-enable-ssh-admin
cli-grant-local-access
cli-undercloud-get-horizon-url
config-download
environment
heat-launcher
outputs
overcloud-deployment_status.yaml
overcloud-export.yaml
overcloud-install-20220823213643.tar.bzip2
overcloud-passwords.yaml
overcloudrc
tempest-deployer-input.conf
tripleo-ansible-inventory.yaml
tripleo-heat-templates
tripleo-overcloud-baremetal-deployment.yaml
tripleo-overcloud-network-data.yaml
tripleo-overcloud-roles-data.yaml
tripleo-overcloud-virtual-ips.yaml
```

The following table describes the content of those files and directories:

Directory	Description
cli-*	Directories used by **ansible-runner** for CLI ansible-based workflows.
config-download	The **config-download** directory, it was previously called~/**config-download** or /**var**/**lib**/**mistral**/**<stack>**.
environment	Contains the saved stack environment generated with the **openstack stack environment show <stack>** command.
heat-launcher	The ephemeral Heat working directory containing the ephemeral Heat configuration and database backups.
outputs	Contains saved stack outputs generated with the **openstack stack output show <stack> <output>** command.
<stack>-deployment_status.yaml	Contains the saved stack status.
<stack>-export.yaml	Contains stack export information, generated with the **openstack overcloud export <stack>** command.
<stack>-install-*.tar.bzip2	A tarball of the working directory.
<stack>-passwords.yaml	Contains the stack passwords.

Directory	Description
<stack>rc	Stack rc credential file required to use the overcloud APIs.
temployer-deployer-input.conf	Contains the Tempest configuration.
tripleo-ansible-inventory.yaml	Ansible inventory for the overcloud.
tripleo-heat-templates	Contains a copy of rendered jinja2 templates. You can find the source templates in **/usr/share/openstack-tripleo-heat-templates** or you can specify the templates with the **--templates** options on the CLI.
tripleo-overcloud-baremetal-deployment.yaml	Baremetal deployment input for provisioning overcloud nodes.
tripleo-overcloud-network-data.yaml	Network deployment input for provisioning overcloud networks.
tripleo-overcloud-roles-data.yaml	Roles data which is specified with the **-r** option on the CLI.
tripleo-overcloud-virtual-ips.yaml	VIP deployment input for provisioning overcloud network VIPs.

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 itself uses OpenStack Platform components in the form of containers to create a toolset called director. This means that the undercloud pulls a set of container images from a registry source, generates configuration for the containers, and runs each OpenStack Platform service as a container. As a result, the undercloud provides a containerized set of services that you can use as a toolset 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.

- 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

Service	Minimum	Maximum
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.

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, as listed on Certified Guest Operating Systems in Red Hat OpenStack Platform, Red Hat Virtualization, OpenShift Virtualization and Red Hat Enterprise Linux with KVM
Red Hat Virtualization	Hosted by Red Hat Virtualization 4.x, as listed on Certified Red Hat Hypervisors.
Microsoft Hyper-V	Hosted by versions of Hyper-V as listed on the Red Hat Customer Portal Certification Catalogue.

Platform	Notes
VMware ESX and ESXi	Hosted by versions of ESX and ESXi as listed on the Red Hat Customer Portal Certification Catalogue.

IMPORTANT

Ensure your hypervisor supports Red Hat Enterprise Linux 9.0 guests.

Virtual machine requirements

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

Network considerations

Power management

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

Provisioning network

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

Allow traffic from an unknown address

You must configure your virtual undercloud hypervisor 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. It is not sufficient to only reboot the VM.

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

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

You run Red Hat OpenStack Platform 17.0 on Red Hat Enterprise Linux 9.0. As a result, you must lock the content from these repositories to the respective Red Hat Enterprise Linux version.

WARNING

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

NOTE

Satellite repositories are not listed because RHOSP 17.0 does not support Satellite. Satellite support is planned for a future release. Only Red Hat CDN is supported as a package repository and container registry.

Core repositories

The following table lists core repositories for installing the undercloud.

Name	Repository	Description of requirement
Red Hat Enterprise Linux 9 for x86_64 – BaseOS (RPMs) Extended Update Support (EUS)	**rhel-9-for-x86_64-baseos-eus-rpms**	Base operating system repository for x86_64 systems.
Red Hat Enterprise Linux 9 for x86_64 – AppStream (RPMs)	**rhel-9-for-x86_64-appstream-eus-rpms**	Contains Red Hat OpenStack Platform dependencies.
Red Hat Enterprise Linux 9 for x86_64 – High Availability (RPMs) Extended Update Support (EUS)	**rhel-9-for-x86_64-highavailability-eus-rpms**	High availability tools for Red Hat Enterprise Linux. Used for Controller node high availability.
Red Hat OpenStack Platform 17.0 for RHEL 9 (RPMs)	**openstack-17-for-rhel-9-x86_64-rpms**	Core Red Hat OpenStack Platform repository, which contains packages for Red Hat OpenStack Platform director.
Red Hat Fast Datapath for RHEL 9 (RPMS)	**fast-datapath-for-rhel-9-x86_64-rpms**	Provides Open vSwitch (OVS) packages for OpenStack Platform.

CHAPTER 3. UNDERSTANDING HEAT TEMPLATES

The custom configurations in this guide use heat templates and environment files to define certain aspects of the overcloud. This chapter provides a basic introduction to heat templates so that you can understand the structure and format of these templates in the context of Red Hat OpenStack Platform director.

3.1. HEAT TEMPLATES

Director uses Heat Orchestration Templates (HOT) as the template format for the overcloud deployment plan. Templates in HOT format are usually expressed in YAML format. The purpose of a template is to define and create a stack, which is a collection of resources that OpenStack Orchestration (heat) creates, and the configuration of the resources. Resources are objects in Red Hat OpenStack Platform (RHOSP) and can include compute resources, network configuration, security groups, scaling rules, and custom resources.

A heat template has three main sections:

parameters

These are settings passed to heat, which provide a way to customize a stack, and any default values for parameters without passed values. These settings are defined in the **parameters** section of a template.

resources

Use the **resources** section to define the resources, such as compute instances, networks, and storage volumes, that you can create when you deploy a stack using this template. Red Hat OpenStack Platform (RHOSP) contains a set of core resources that span across all components. These are the specific objects to create and configure as part of a stack. RHOSP contains a set of core resources that span across all components. These are defined in the **resources** section of a template.

outputs

Use the **outputs** section to declare the output parameters that your cloud users can access after the stack is created. Your cloud users can use these parameters to request details about the stack, such as the IP addresses of deployed instances, or URLs of web applications deployed as part of the stack.

Example of a basic heat template:

```
heat_template_version: 2013-05-23

description: > A very basic Heat template.

parameters:
  key_name:
    type: string
    default: lars
    description: Name of an existing key pair to use for the instance
  flavor:
    type: string
    description: Instance type for the instance to be created
    default: m1.small
  image:
    type: string
    default: cirros
    description: ID or name of the image to use for the instance
```

```
resources:
  my_instance:
    type: OS::Nova::Server
    properties:
      name: My Cirros Instance
      image: { get_param: image }
      flavor: { get_param: flavor }
      key_name: { get_param: key_name }

output:
  instance_name:
    description: Get the instance's name
    value: { get_attr: [ my_instance, name ] }
```

This template uses the resource type **type: OS::Nova::Server** to create an instance called **my_instance** with a particular flavor, image, and key that the cloud user specifies. The stack can return the value of **instance_name**, which is called **My Cirros Instance**.

When heat processes a template, it creates a stack for the template and a set of child stacks for resource templates. This creates a hierarchy of stacks that descend from the main stack that you define with your template. You can view the stack hierarchy with the following command:

```
$ openstack stack list --nested
```

3.2. ENVIRONMENT FILES

An environment file is a special type of template that you can use to customize your heat templates. You can include environment files in the deployment command, in addition to the core heat templates. An environment file contains three main sections:

resource_registry

This section defines custom resource names, linked to other heat templates. This provides a method to create custom resources that do not exist within the core resource collection.

parameters

These are common settings that you apply to the parameters of the top-level template. For example, if you have a template that deploys nested stacks, such as resource registry mappings, the parameters apply only to the top-level template and not to templates for the nested resources.

parameter_defaults

These parameters modify the default values for parameters in all templates. For example, if you have a heat template that deploys nested stacks, such as resource registry mappings,the parameter defaults apply to all templates.

IMPORTANT

Use **parameter_defaults** instead of **parameters** when you create custom environment files for your overcloud, so that your parameters apply to all stack templates for the overcloud.

Example of a basic environment file:

```
resource_registry:
  OS::Nova::Server::MyServer: myserver.yaml

parameter_defaults:
  NetworkName: my_network

parameters:
  MyIP: 192.168.0.1
```

This environment file (**my_env.yaml**) might be included when creating a stack from a certain heat template (**my_template.yaml**). The **my_env.yaml** file creates a new resource type called **OS::Nova::Server::MyServer**. The **myserver.yaml** file is a heat template file that provides an implementation for this resource type that overrides any built-in ones. You can include the **OS::Nova::Server::MyServer** resource in your **my_template.yaml** file.

MyIP applies a parameter only to the main heat template that deploys with this environment file. In this example, **MyIP** applies only to the parameters in **my_template.yaml**.

NetworkName applies to both the main heat template, **my_template.yaml**, and the templates that are associated with the resources that are included in the main template, such as the **OS::Nova::Server::MyServer** resource and its **myserver.yaml** template in this example.

NOTE

For RHOSP to use the heat template file as a custom template resource, the file extension must be either .yaml or .template.

3.3. CORE OVERCLOUD HEAT TEMPLATES

Director contains a core heat template collection and environment file collection for the overcloud. This collection is stored in **/usr/share/openstack-tripleo-heat-templates**.

The main files and directories in this template collection are:

overcloud.j2.yaml

This is the main template file that director uses to create the overcloud environment. This file uses Jinja2 syntax to iterate over certain sections in the template to create custom roles. The Jinja2 formatting is rendered into YAML during the overcloud deployment process.

overcloud-resource-registry-puppet.j2.yaml

This is the main environment file that director uses to create the overcloud environment. It provides a set of configurations for Puppet modules stored on the overcloud image. After director writes the overcloud image to each node, heat starts the Puppet configuration for each node by using the resources registered in this environment file. This file uses Jinja2 syntax to iterate over certain sections in the template to create custom roles. The Jinja2 formatting is rendered into YAML during the overcloud deployment process.

roles_data.yaml

This file contains the definitions of the roles in an overcloud and maps services to each role.

network_data.yaml

This file contains the definitions of the networks in an overcloud and their properties such as subnets, allocation pools, and VIP status. The default **network_data.yaml** file contains the default networks: External, Internal Api, Storage, Storage Management, Tenant, and Management. You can create a custom **network_data.yaml** file and add it to your **openstack overcloud deploy** command with the **-n** option.

plan-environment.yaml

This file contains the definitions of the metadata for your overcloud plan. This includes the plan name, main template to use, and environment files to apply to the overcloud.

capabilities-map.yaml

This file contains a mapping of environment files for an overcloud plan.

deployment

This directory contains heat templates. The **overcloud-resource-registry-puppet.j2.yaml** environment file uses the files in this directory to drive the application of the Puppet configuration on each node.

environments

This directory contains additional heat environment files that you can use for your overcloud creation. These environment files enable extra functions for your resulting Red Hat OpenStack Platform (RHOSP) environment. For example, the directory contains an environment file to enable Cinder NetApp backend storage (**cinder-netapp-config.yaml**).

network

This directory contains a set of heat templates that you can use to create isolated networks and ports.

puppet

This directory contains templates that control Puppet configuration. The **overcloud-resource-registry-puppet.j2.yaml** environment file uses the files in this directory to drive the application of the Puppet configuration on each node.

puppet/services

This directory contains legacy heat templates for all service configuration. The templates in the **deployment** directory replace most of the templates in the **puppet/services** directory.

extraconfig

This directory contains templates that you can use to enable extra functionality.

3.4. PLAN ENVIRONMENT METADATA

You can define metadata for your overcloud plan in a plan environment metadata file. Director applies metadata during the overcloud creation, and when importing and exporting your overcloud plan.

A plan environment metadata file includes the following parameters:

version

The version of the template.

name

The name of the overcloud plan and the container in OpenStack Object Storage (swift) that you want to use to store the plan files.

template

The core parent template that you want to use for the overcloud deployment. This is most often **overcloud.yaml**, which is the rendered version of the **overcloud.yaml.j2** template.

environments

Defines a list of environment files that you want to use. Specify the name and relative locations of each environment file with the **path** sub-parameter.

parameter_defaults

A set of parameters that you want to use in your overcloud. This functions in the same way as the **parameter_defaults** section in a standard environment file.

passwords

A set of parameters that you want to use for overcloud passwords. This functions in the same way as the **parameter_defaults** section in a standard environment file. Usually, the director populates this section automatically with randomly generated passwords.

The following snippet is an example of the syntax of a plan environment file:

```
version: 1.0
name: myovercloud
description: 'My Overcloud Plan'
template: overcloud.yaml
environments:
- path: overcloud-resource-registry-puppet.yaml
- path: environments/containers-default-parameters.yaml
- path: user-environment.yaml
parameter_defaults:
  ControllerCount: 1
  ComputeCount: 1
  OvercloudComputeFlavor: compute
  OvercloudControllerFlavor: control
workflow_parameters:
  tripleo.derive_params.v1.derive_parameters:
    num_phy_cores_per_numa_node_for_pmd: 2
```

You can include the plan environment metadata file with the **openstack overcloud deploy** command with the **-p** option:

```
(undercloud) $ openstack overcloud deploy --templates \
  -p /my-plan-environment.yaml \
  [OTHER OPTIONS]
```

You can also view plan metadata for an existing overcloud plan with the following command:

```
(undercloud) $ openstack object save overcloud plan-environment.yaml --file -
```

3.5. INCLUDING ENVIRONMENT FILES IN OVERCLOUD CREATION

Include environment files in the deployment command with the **-e** option. You can include as many environment files as necessary. However, the order of the environment files is important as the parameters and resources that you define in subsequent environment files take precedence. For example, you have two environment files that contain a common resource type **OS::TripleO::NodeExtraConfigPost**, and a common parameter **TimeZone**:

environment-file-1.yaml

```
resource_registry:
  OS::TripleO::NodeExtraConfigPost: /home/stack/templates/template-1.yaml

parameter_defaults:
  RabbitFDLimit: 65536
  TimeZone: 'Japan'
```

environment-file-2.yaml

```
resource_registry:
  OS::TripleO::NodeExtraConfigPost: /home/stack/templates/template-2.yaml

parameter_defaults:
  TimeZone: 'Hongkong'
```

You include both environment files in the deployment command:

```
$ openstack overcloud deploy --templates -e environment-file-1.yaml -e environment-file-2.yaml
```

The **openstack overcloud deploy** command runs through the following process:

1. Loads the default configuration from the core heat template collection.
2. Applies the configuration from **environment-file-1.yaml**, which overrides any common settings from the default configuration.
3. Applies the configuration from **environment-file-2.yaml**, which overrides any common settings from the default configuration and **environment-file-1.yaml**.

This results in the following changes to the default configuration of the overcloud:

- **OS::TripleO::NodeExtraConfigPost** resource is set to **/home/stack/templates/template-2.yaml**, as defined in **environment-file-2.yaml**.
- **TimeZone** parameter is set to **Hongkong**, as defined in **environment-file-2.yaml**.
- **RabbitFDLimit** parameter is set to **65536**, as defined in **environment-file-1.yaml**. **environment-file-2.yaml** does not change this value.

You can use this mechanism to define custom configuration for your overcloud without values from multiple environment files conflicting.

3.6. USING CUSTOMIZED CORE HEAT TEMPLATES

When creating the overcloud, director uses a core set of heat templates located in **/usr/share/openstack-tripleo-heat-templates**. If you want to customize this core template collection, use the following Git workflows to manage your custom template collection:

Procedure

- Create an initial Git repository that contains the heat template collection:
  a. Copy the template collection to the **/home/stack/templates** directory:

  ```
  $ cd ~/templates
  $ cp -r /usr/share/openstack-tripleo-heat-templates .
  ```

  b. Change to the custom template directory and initialize a Git repository:

  ```
  $ cd ~/templates/openstack-tripleo-heat-templates
  $ git init .
  ```

  c. Configure your Git user name and email address:

```
$ git config --global user.name "<USER_NAME>"
$ git config --global user.email "<EMAIL_ADDRESS>"
```

- Replace **<USER_NAME>** with the user name that you want to use.
- Replace **<EMAIL_ADDRESS>** with your email address.

  a. Stage all templates for the initial commit:

  ```
  $ git add *
  ```

  b. Create an initial commit:

  ```
  $ git commit -m "Initial creation of custom core heat templates"
  ```

  This creates an initial **master** branch that contains the latest core template collection. Use this branch as the basis for your custom branch and merge new template versions to this branch.

- Use a custom branch to store your changes to the core template collection. Use the following procedure to create a **my-customizations** branch and add customizations:

  a. Create the **my-customizations** branch and switch to it:

  ```
  $ git checkout -b my-customizations
  ```

  b. Edit the files in the custom branch.

  c. Stage the changes in git:

  ```
  $ git add [edited files]
  ```

  d. Commit the changes to the custom branch:

  ```
  $ git commit -m "[Commit message for custom changes]"
  ```

  This adds your changes as commits to the **my-customizations** branch. When the **master** branch updates, you can rebase **my-customizations** off **master**, which causes git to add these commits on to the updated template collection. This helps track your customizations and replay them on future template updates.

- When you update the undercloud, the **openstack-tripleo-heat-templates** package might also receive updates. When this occurs, you must also update your custom template collection:

  a. Save the **openstack-tripleo-heat-templates** package version as an environment variable:

  ```
  $ export PACKAGE=$(rpm -qv openstack-tripleo-heat-templates)
  ```

  b. Change to your template collection directory and create a new branch for the updated templates:

  ```
  $ cd ~/templates/openstack-tripleo-heat-templates
  $ git checkout -b $PACKAGE
  ```

c. Remove all files in the branch and replace them with the new versions:

```
$ git rm -rf *
$ cp -r /usr/share/openstack-tripleo-heat-templates/* .
```

d. Add all templates for the initial commit:

```
$ git add *
```

e. Create a commit for the package update:

```
$ git commit -m "Updates for $PACKAGE"
```

f. Merge the branch into master. If you use a Git management system (such as GitLab), use the management workflow. If you use git locally, merge by switching to the **master** branch and run the **git merge** command:

```
$ git checkout master
$ git merge $PACKAGE
```

The **master** branch now contains the latest version of the core template collection. You can now rebase the **my-customization** branch from this updated collection.

- Update the **my-customization** branch,:

  a. Change to the **my-customizations** branch:

  ```
  $ git checkout my-customizations
  ```

  b. Rebase the branch off **master**:

  ```
  $ git rebase master
  ```

  This updates the **my-customizations** branch and replays the custom commits made to this branch.

- Resolve any conflicts that occur during the rebase:

  a. Check which files contain the conflicts:

  ```
  $ git status
  ```

  b. Resolve the conflicts of the template files identified.

  c. Add the resolved files:

  ```
  $ git add [resolved files]
  ```

  d. Continue the rebase:

  ```
  $ git rebase --continue
  ```

- Deploy the custom template collection:

a. Ensure that you have switched to the **my-customization** branch:

```
git checkout my-customizations
```

b. Run the **openstack overcloud deploy** command with the **--templates** option to specify your local template directory:

```
$ openstack overcloud deploy --templates /home/stack/templates/openstack-tripleo-heat-templates [OTHER OPTIONS]
```

NOTE

Director uses the default template directory (**/usr/share/openstack-tripleo-heat-templates**) if you specify the **--templates** option without a directory.

IMPORTANT

Red Hat recommends using the methods in Chapter 5, *Configuration hooks* instead of modifying the heat template collection.

3.7. JINJA2 RENDERING

The core heat templates in **/usr/share/openstack-tripleo-heat-templates** contain a number of files that have the **j2.yaml** file extension. These files contain Jinja2 template syntax and director renders these files to their static heat template equivalents that have the **.yaml** extension. For example, the main **overcloud.j2.yaml** file renders into **overcloud.yaml**. Director uses the resulting **overcloud.yaml** file.

The Jinja2-enabled heat templates use Jinja2 syntax to create parameters and resources for iterative values. For example, the **overcloud.j2.yaml** file contains the following snippet:

```
parameters:
...
{% for role in roles %}
  ...
  {{role.name}}Count:
    description: Number of {{role.name}} nodes to deploy
    type: number
    default: {{role.CountDefault|default(0)}}
  ...
{% endfor %}
```

When director renders the Jinja2 syntax, director iterates over the roles defined in the **roles_data.yaml** file and populates the **{{role.name}}Count** parameter with the name of the role. The default **roles_data.yaml** file contains five roles and results in the following parameters from our example:

- **ControllerCount**
- **ComputeCount**
- **BlockStorageCount**
- **ObjectStorageCount**
- **CephStorageCount**

A example rendered version of the parameter looks like this:

```
parameters:
  ...
  ControllerCount:
    description: Number of Controller nodes to deploy
    type: number
    default: 1
  ...
```

Director renders Jinja2-enabled templates and environment files only from within the directory of your core heat templates. The following use cases demonstrate the correct method to render the Jinja2 templates.

Use case 1: Default core templates

Template directory: /usr/share/openstack-tripleo-heat-templates/

Environment file: /usr/share/openstack-tripleo-heat-templates/environments/ssl/enable-internal-tls.j2.yaml

Director uses the default core template location (**--templates**) and renders the **enable-internal-tls.j2.yaml** file into **enable-internal-tls.yaml**. When you run the **openstack overcloud deploy** command, use the **-e** option to include the name of the rendered **enable-internal-tls.yaml** file.

```
$ openstack overcloud deploy --templates \
    -e /usr/share/openstack-tripleo-heat-templates/environments/ssl/enable-internal-tls.yaml
    ...
```

Use case 2: Custom core templates

Template directory: /home/stack/tripleo-heat-installer-templates

Environment file: /home/stack/tripleo-heat-installer-templates/environments/ssl/enable-internal-tls.j2.yaml

Director uses a custom core template location (**--templates /home/stack/tripleo-heat-templates**) and director renders the **enable-internal-tls.j2.yaml** file within the custom core templates into **enable-internal-tls.yaml**. When you run the **openstack overcloud deploy** command, use the **-e** option to include the name of the rendered **enable-internal-tls.yaml** file.

```
$ openstack overcloud deploy --templates /home/stack/tripleo-heat-templates \
    -e /home/stack/tripleo-heat-templates/environments/ssl/enable-internal-tls.yaml
    ...
```

Use case 3: Incorrect usage

Template directory: /usr/share/openstack-tripleo-heat-templates/

Environment file: /home/stack/tripleo-heat-installer-templates/environments/ssl/enable-internal-tls.j2.yaml

Director uses a custom core template location (**--templates /home/stack/tripleo-heat-installer-templates**). However, the chosen **enable-internal-tls.j2.yaml** is not located within the custom core templates, so it will not render into **enable-internal-tls.yaml**. This causes the deployment to fail.

Processing Jinja2 syntax into static templates

Use the **process-templates.py** script to render the Jinja2 syntax of the **openstack-tripleo-heat-templates** into a set of static templates. To render a copy of the **openstack-tripleo-heat-templates** collection with the **process-templates.py** script, change to the **openstack-tripleo-heat-templates** directory:

```
$ cd /usr/share/openstack-tripleo-heat-templates
```

Run the **process-templates.py** script, which is located in the **tools** directory, along with the **-o** option to define a custom directory to save the static copy:

```
$ ./tools/process-templates.py -o ~/openstack-tripleo-heat-templates-rendered
```

This converts all Jinja2 templates to their rendered YAML versions and saves the results to **~/openstack-tripleo-heat-templates-rendered**.

CHAPTER 4. HEAT PARAMETERS

Each heat template in the director template collection contains a **parameters** section. This section contains definitions for all parameters specific to a particular overcloud service. This includes the following:

- **overcloud.j2.yaml** - Default base parameters
- **roles_data.yaml** - Default parameters for composable roles
- **deployment/*.yaml** - Default parameters for specific services

You can modify the values for these parameters using the following method:

1. Create an environment file for your custom parameters.
2. Include your custom parameters in the **parameter_defaults** section of the environment file.
3. Include the environment file with the **openstack overcloud deploy** command.

4.1. EXAMPLE 1: CONFIGURING THE TIME ZONE

The Heat template for setting the timezone (**puppet/services/time/timezone.yaml**) contains a **TimeZone** parameter. If you leave the **TimeZone** parameter blank, the overcloud sets the time to **UTC** as a default.

To obtain lists of timezones run the **timedatectl list-timezones** command. The following example command retrieves the timezones for Asia:

```
$ sudo timedatectl list-timezones|grep "Asia"
```

After you identify your timezone, set the *TimeZone* parameter in an environment file. The following example environment file sets the value of *TimeZone* to *Asia/Tokyo*:

```
parameter_defaults:
  TimeZone: 'Asia/Tokyo'
```

4.2. EXAMPLE 2: CONFIGURING RABBITMQ FILE DESCRIPTOR LIMIT

For certain configurations, you might need to increase the file descriptor limit for the RabbitMQ server. Use the **deployment/rabbitmq/rabbitmq-container-puppet.yaml** heat template to set a new limit in the **RabbitFDLimit** parameter. Add the following entry to an environment file:

```
parameter_defaults:
  RabbitFDLimit: 65536
```

4.3. EXAMPLE 3: ENABLING AND DISABLING PARAMETERS

You might need to initially set a parameter during a deployment, then disable the parameter for a future deployment operation, such as updates or scaling operations. For example, to include a custom RPM during the overcloud creation, include the following entry in an environment file:

```
parameter_defaults:
  DeployArtifactURLs: ["http://www.example.com/myfile.rpm"]
```

To disable this parameter from a future deployment, it is not sufficient to remove the parameter. Instead, you must set the parameter to an empty value:

```
parameter_defaults:
  DeployArtifactURLs: []
```

This ensures the parameter is no longer set for subsequent deployments operations.

4.4. EXAMPLE 4: ROLE-BASED PARAMETERS

Use the **[ROLE]Parameters** parameters, replacing **[ROLE]** with a composable role, to set parameters for a specific role.

For example, director configures **sshd** on both Controller and Compute nodes. To set a different **sshd** parameters for Controller and Compute nodes, create an environment file that contains both the **ControllerParameters** and **ComputeParameters** parameter and set the **sshd** parameters for each specific role:

```
parameter_defaults:
  ControllerParameters:
    BannerText: "This is a Controller node"
  ComputeParameters:
    BannerText: "This is a Compute node"
```

4.5. IDENTIFYING PARAMETERS THAT YOU WANT TO MODIFY

Red Hat OpenStack Platform director provides many parameters for configuration. In some cases, you might experience difficulty identifying a certain option that you want to configure, and the corresponding director parameter. If there is an option that you want to configure with director, use the following workflow to identify and map the option to a specific overcloud parameter:

1. Identify the option that you want to configure. Make a note of the service that uses the option.
2. Check the corresponding Puppet module for this option. The Puppet modules for Red Hat OpenStack Platform are located under **/etc/puppet/modules** on the director node. Each module corresponds to a particular service. For example, the **keystone** module corresponds to the OpenStack Identity (keystone).
    - If the Puppet module contains a variable that controls the chosen option, move to the next step.
    - If the Puppet module does not contain a variable that controls the chosen option, no hieradata exists for this option. If possible, you can set the option manually after the overcloud completes deployment.
3. Check the core heat template collection for the Puppet variable in the form of hieradata. The templates in **deployment/*** usually correspond to the Puppet modules of the same services. For example, the **deployment/keystone/keystone-container-puppet.yaml** template provides hieradata to the **keystone** module.

- If the heat template sets hieradata for the Puppet variable, the template should also disclose the director-based parameter that you can modify.
- If the heat template does not set hieradata for the Puppet variable, use the configuration hooks to pass the hieradata using an environment file. See Section 5.4, “Puppet: Customizing hieradata for roles” for more information on customizing hieradata.

Procedure

1. To change the notification format for OpenStack Identity (keystone), use the workflow and complete the following steps:

   a. Identify the OpenStack parameter that you want to configure (**notification_format**).

   b. Search the **keystone** Puppet module for the **notification_format** setting:

   ```
   $ grep notification_format /etc/puppet/modules/keystone/manifests/*
   ```

   In this case, the **keystone** module manages this option using the **keystone::notification_format** variable.

   c. Search the **keystone** service template for this variable:

   ```
   $ grep "keystone::notification_format" /usr/share/openstack-tripleo-heat-
   templates/deployment/keystone/keystone-container-puppet.yaml
   ```

   The output shows that director uses the **KeystoneNotificationFormat** parameter to set the **keystone::notification_format** hieradata.

The following table shows the eventual mapping:

Director parameter	Puppet hieradata	OpenStack Identity (keystone) option
KeystoneNotificationFormat	**keystone::notification_format**	**notification_format**

You set the **KeystoneNotificationFormat** in an overcloud environment file, which then sets the **notification_format** option in the **keystone.conf** file during the overcloud configuration.

CHAPTER 5. CONFIGURATION HOOKS

Use configuration hooks to inject your own custom configuration functions into the overcloud deployment process. You can create hooks to inject custom configuration before and after the main overcloud services configuration, and hooks for modifying and including Puppet-based configuration.

5.1. PRE-CONFIGURATION: CUSTOMIZING SPECIFIC OVERCLOUD ROLES

The overcloud uses Puppet for the core configuration of OpenStack components. Director provides a set of hooks that you can use to perform custom configuration for specific node roles before the core configuration begins. These hooks include the following configurations:

IMPORTANT

Previous versions of this document used the **OS::TripleO::Tasks::*PreConfig** resources to provide pre-configuration hooks on a per role basis. The heat template collection requires dedicated use of these hooks, which means that you should not use them for custom use. Instead, use the **OS::TripleO::*ExtraConfigPre** hooks outlined here.

OS::TripleO::ControllerExtraConfigPre
: Additional configuration applied to Controller nodes before the core Puppet configuration.

OS::TripleO::ComputeExtraConfigPre
: Additional configuration applied to Compute nodes before the core Puppet configuration.

OS::TripleO::CephStorageExtraConfigPre
: Additional configuration applied to Ceph Storage nodes before the core Puppet configuration.

OS::TripleO::ObjectStorageExtraConfigPre
: Additional configuration applied to Object Storage nodes before the core Puppet configuration.

OS::TripleO::BlockStorageExtraConfigPre
: Additional configuration applied to Block Storage nodes before the core Puppet configuration.

OS::TripleO::[ROLE]ExtraConfigPre
: Additional configuration applied to custom nodes before the core Puppet configuration. Replace **[ROLE]** with the composable role name.

In this example, append the **resolv.conf** file on all nodes of a particular role with a variable nameserver:

Procedure

1. Create a basic heat template **~/templates/nameserver.yaml** that runs a script to write a variable nameserver to the **resolv.conf** file of a node:

```
heat_template_version: 2014-10-16

description: >
  Extra hostname configuration

parameters:
  server:
    type: string
  nameserver_ip:
    type: string
  DeployIdentifier:
    type: string

resources:
  CustomExtraConfigPre:
    type: OS::Heat::SoftwareConfig
    properties:
      group: script
      config:
        str_replace:
          template: |
            #!/bin/sh
            echo "nameserver _NAMESERVER_IP_" > /etc/resolv.conf
          params:
            _NAMESERVER_IP_: {get_param: nameserver_ip}

  CustomExtraDeploymentPre:
    type: OS::Heat::SoftwareDeployment
    properties:
      server: {get_param: server}
      config: {get_resource: CustomExtraConfigPre}
      actions: ['CREATE']
      input_values:
        deploy_identifier: {get_param: DeployIdentifier}

outputs:
  deploy_stdout:
    description: Deployment reference, used to trigger pre-deploy on changes
    value: {get_attr: [CustomExtraDeploymentPre, deploy_stdout]}
```

In this example, the **resources** section contains the following parameters:

CustomExtraConfigPre

This defines a software configuration. In this example, we define a Bash **script** and heat replaces **_NAMESERVER_IP_** with the value stored in the **nameserver_ip** parameter.

CustomExtraDeploymentPre

This executes a software configuration, which is the software configuration from the **CustomExtraConfigPre** resource. Note the following:

- The **config** parameter references the **CustomExtraConfigPre** resource so that heat knows which configuration to apply.
- The **server** parameter retrieves a map of the overcloud nodes. This parameter is provided by the parent template and is mandatory in templates for this hook.
- The **actions** parameter defines when to apply the configuration. In this case, you want to apply the configuration when the overcloud is created. Possible actions include **CREATE**, **UPDATE**, **DELETE**, **SUSPEND**, and **RESUME**.
- **input_values** contains a parameter called **deploy_identifier**, which stores the **DeployIdentifier** from the parent template. This parameter provides a timestamp to the resource for each deployment update to ensure that the resource reapplies on subsequent overcloud updates.

2. Create an environment file **~/templates/pre_config.yaml** that registers your heat template to the role-based resource type. For example, to apply the configuration only to Controller nodes, use the **ControllerExtraConfigPre** hook:

```
resource_registry:
  OS::TripleO::ControllerExtraConfigPre: /home/stack/templates/nameserver.yaml

parameter_defaults:
  nameserver_ip: 192.168.1.1
```

3. Add the environment file to the stack, along with your other environment files:

```
$ openstack overcloud deploy --templates \
    ...
    -e /home/stack/templates/pre_config.yaml \
    ...
```

   This applies the configuration to all Controller nodes before the core configuration begins on either the initial overcloud creation or subsequent updates.

IMPORTANT

You can register each resource to only one heat template per hook. Subsequent usage overrides the heat template to use.

5.2. PRE-CONFIGURATION: CUSTOMIZING ALL OVERCLOUD ROLES

The overcloud uses Puppet for the core configuration of OpenStack components. Director provides a hook that you can use to configure all node types before the core configuration begins:

OS::TripleO::NodeExtraConfig
: Additional configuration applied to all nodes roles before the core Puppet configuration.

In this example, append the **resolv.conf** file on each node with a variable nameserver:

Procedure

1. Create a basic heat template **~/templates/nameserver.yaml** that runs a script to append the **resolv.conf** file of each node with a variable nameserver:

```
heat_template_version: 2014-10-16

description: >
  Extra hostname configuration

parameters:
  server:
    type: string
  nameserver_ip:
    type: string
  DeployIdentifier:
    type: string

resources:
  CustomExtraConfigPre:
    type: OS::Heat::SoftwareConfig
    properties:
      group: script
      config:
        str_replace:
          template: |
            #!/bin/sh
            echo "nameserver _NAMESERVER_IP_" >> /etc/resolv.conf
          params:
            _NAMESERVER_IP_: {get_param: nameserver_ip}

  CustomExtraDeploymentPre:
    type: OS::Heat::SoftwareDeployment
    properties:
      server: {get_param: server}
      config: {get_resource: CustomExtraConfigPre}
      actions: ['CREATE']
      input_values:
        deploy_identifier: {get_param: DeployIdentifier}

outputs:
  deploy_stdout:
    description: Deployment reference, used to trigger pre-deploy on changes
    value: {get_attr: [CustomExtraDeploymentPre, deploy_stdout]}
```

In this example, the **resources** section contains the following parameters:

CustomExtraConfigPre

This parameter defines a software configuration. In this example, you define a Bash **script** and heat replaces **_NAMESERVER_IP_** with the value stored in the **nameserver_ip** parameter.

CustomExtraDeploymentPre

This parameter executes a software configuration, which is the software configuration from the **CustomExtraConfigPre** resource. Note the following:

- The **config** parameter references the **CustomExtraConfigPre** resource so that heat knows which configuration to apply.
- The **server** parameter retrieves a map of the overcloud nodes. This parameter is provided by the parent template and is mandatory in templates for this hook.
- The **actions** parameter defines when to apply the configuration. In this case, you only apply the configuration when the overcloud is created. Possible actions include **CREATE**, **UPDATE**, **DELETE**, **SUSPEND**, and **RESUME**.
- The **input_values** parameter contains a sub-parameter called **deploy_identifier**, which stores the **DeployIdentifier** from the parent template. This parameter provides a timestamp to the resource for each deployment update to ensure that the resource reapplies on subsequent overcloud updates.

2. Create an environment file **~/templates/pre_config.yaml** that registers your heat template as the **OS::TripleO::NodeExtraConfig** resource type.

```
resource_registry:
  OS::TripleO::NodeExtraConfig: /home/stack/templates/nameserver.yaml

parameter_defaults:
  nameserver_ip: 192.168.1.1
```

3. Add the environment file to the stack, along with your other environment files:

```
$ openstack overcloud deploy --templates \
    ...
    -e /home/stack/templates/pre_config.yaml \
    ...
```

This applies the configuration to all nodes before the core configuration begins on either the initial overcloud creation or subsequent updates.

IMPORTANT

You can register the **OS::TripleO::NodeExtraConfig** to only one heat template. Subsequent usage overrides the heat template to use.

5.3. POST-CONFIGURATION: CUSTOMIZING ALL OVERCLOUD ROLES

IMPORTANT

Previous versions of this document used the **OS::TripleO::Tasks::*PostConfig** resources to provide post-configuration hooks on a per role basis. The heat template collection requires dedicated use of these hooks, which means that you should not use them for custom use. Instead, use the **OS::TripleO::NodeExtraConfigPost** hook outlined here.

A situation might occur where you have completed the creation of your overcloud but you want to add additional configuration to all roles, either on initial creation or on a subsequent update of the overcloud. In this case, use the following post-configuration hook:

OS::TripleO::NodeExtraConfigPost
: Additional configuration applied to all nodes roles after the core Puppet configuration.

In this example, append the **resolv.conf** file on each node with a variable nameserver:

Procedure

1. Create a basic heat template **~/templates/nameserver.yaml** that runs a script to append the **resolv.conf** file of each node with a variable nameserver:

```
heat_template_version: 2014-10-16

description: >
  Extra hostname configuration

parameters:
  servers:
    type: json
  nameserver_ip:
    type: string
  DeployIdentifier:
    type: string
  EndpointMap:
    default: {}
    type: json

resources:
  CustomExtraConfig:
    type: OS::Heat::SoftwareConfig
    properties:
      group: script
      config:
        str_replace:
          template: |
            #!/bin/sh
            echo "nameserver _NAMESERVER_IP_" >> /etc/resolv.conf
          params:
            _NAMESERVER_IP_: {get_param: nameserver_ip}

  CustomExtraDeployments:
    type: OS::Heat::SoftwareDeploymentGroup
    properties:
      servers:  {get_param: servers}
      config: {get_resource: CustomExtraConfig}
      actions: ['CREATE']
      input_values:
        deploy_identifier: {get_param: DeployIdentifier}
```

In this example, the **resources** section contains the following parameters:

CustomExtraConfig

This defines a software configuration. In this example, you define a Bash **script** and heat replaces **_NAMESERVER_IP_** with the value stored in the **nameserver_ip** parameter.

CustomExtraDeployments

This executes a software configuration, which is the software configuration from the **CustomExtraConfig** resource. Note the following:

- The **config** parameter references the **CustomExtraConfig** resource so that heat knows which configuration to apply.
- The **servers** parameter retrieves a map of the overcloud nodes. This parameter is provided by the parent template and is mandatory in templates for this hook.
- The **actions** parameter defines when to apply the configuration. In this case, you want apply the configuration when the overcloud is created. Possible actions include **CREATE**, **UPDATE**, **DELETE**, **SUSPEND**, and **RESUME**.
- **input_values** contains a parameter called **deploy_identifier**, which stores the **DeployIdentifier** from the parent template. This parameter provides a timestamp to the resource for each deployment update to ensure that the resource reapplies on subsequent overcloud updates.

2. Create an environment file **~/templates/post_config.yaml** that registers your heat template as the **OS::TripleO::NodeExtraConfigPost:** resource type.

```
resource_registry:
  OS::TripleO::NodeExtraConfigPost: /home/stack/templates/nameserver.yaml

parameter_defaults:
  nameserver_ip: 192.168.1.1
```

3. Add the environment file to the stack, along with your other environment files:

```
$ openstack overcloud deploy --templates \
    ...
    -e /home/stack/templates/post_config.yaml \
    ...
```

   This applies the configuration to all nodes after the core configuration completes on either initial overcloud creation or subsequent updates.

IMPORTANT

You can register the **OS::TripleO::NodeExtraConfigPost** to only one heat template. Subsequent usage overrides the heat template to use.

5.4. PUPPET: CUSTOMIZING HIERADATA FOR ROLES

The heat template collection contains a set of parameters that you can use to pass extra configuration to certain node types. These parameters save the configuration as hieradata for the Puppet configuration on the node:

ControllerExtraConfig
: Configuration to add to all Controller nodes.

ComputeExtraConfig
: Configuration to add to all Compute nodes.

BlockStorageExtraConfig
: Configuration to add to all Block Storage nodes.

ObjectStorageExtraConfig
: Configuration to add to all Object Storage nodes.

CephStorageExtraConfig
: Configuration to add to all Ceph Storage nodes.

[ROLE]ExtraConfig
: Configuration to add to a composable role. Replace **[ROLE]** with the composable role name.

ExtraConfig
: Configuration to add to all nodes.

Procedure

1. To add extra configuration to the post-deployment configuration process, create an environment file that contains these parameters in the **parameter_defaults** section. For example, to increase the reserved memory for Compute hosts to 1024 MB and set the VNC

keymap to Japanese, use the following entries in the **ComputeExtraConfig** parameter:

```
parameter_defaults:
  ComputeExtraConfig:
    nova::compute::reserved_host_memory: 1024
    nova::compute::vnc_keymap: ja
```

2. Include this environment file in the **openstack overcloud deploy** command, along with any other environment files relevant to your deployment.

IMPORTANT

You can define each parameter only once. Subsequent usage overrides previous values.

5.5. PUPPET: CUSTOMIZING HIERADATA FOR INDIVIDUAL NODES

You can set Puppet hieradata for individual nodes using the heat template collection:

Procedure

1. Identify the system UUID from the introspection data for a node:

```
$ openstack baremetal introspection data save 9dcc87ae-4c6d-4ede-81a5-9b20d7dc4a14 |
jq .extra.system.product.uuid
```

This command returns a system UUID. For example:

```
"f5055c6c-477f-47fb-afe5-95c6928c407f"
```

2. Create an environment file to define node-specific hieradata and register the **per_node.yaml** template to a pre-configuration hook. Include the system UUID of the node that you want to configure in the **NodeDataLookup** parameter:

```
resource_registry:
  OS::TripleO::ComputeExtraConfigPre: /usr/share/openstack-tripleo-heat-
templates/puppet/extraconfig/pre_deploy/per_node.yaml
parameter_defaults:
  NodeDataLookup: '{"f5055c6c-477f-47fb-afe5-95c6928c407f":
{"nova::compute::vcpu_pin_set": [ "2", "3" ]}}'
```

3. Include this environment file in the **openstack overcloud deploy** command, along with any other environment files relevant to your deployment.

The **per_node.yaml** template generates a set of hieradata files on nodes that correspond to each system UUID and contains the hieradata that you define. If a UUID is not defined, the resulting hieradata file is empty. In this example, the **per_node.yaml** template runs on all Compute nodes as defined by the **OS::TripleO::ComputeExtraConfigPre** hook, but only the Compute node with system UUID **f5055c6c-477f-47fb-afe5-95c6928c407f** receives hieradata.

You can use this mechanism to tailor each node according to specific requirements.

5.6. PUPPET: APPLYING CUSTOM MANIFESTS

In certain circumstances, you might want to install and configure some additional components on your overcloud nodes. You can achieve this with a custom Puppet manifest that applies to nodes after the main configuration completes. As a basic example, you might want to install **motd** on each node

Procedure

1. Create a heat template **~/templates/custom_puppet_config.yaml** that launches Puppet configuration.

   ```
   heat_template_version: 2014-10-16

   description: >
     Run Puppet extra configuration to set new MOTD

   parameters:
     servers:
       type: json
     DeployIdentifier:
       type: string
     EndpointMap:
       default: {}
       type: json

   resources:
     ExtraPuppetConfig:
       type: OS::Heat::SoftwareConfig
       properties:
         config: {get_file: motd.pp}
         group: puppet
         options:
           enable_hiera: True
           enable_facter: False

     ExtraPuppetDeployments:
       type: OS::Heat::SoftwareDeploymentGroup
       properties:
         config: {get_resource: ExtraPuppetConfig}
         servers: {get_param: servers}
   ```

   This example includes the **/home/stack/templates/motd.pp** within the template and passes it to nodes for configuration. The **motd.pp** file contains the Puppet classes necessary to install and configure **motd**.

2. Create an environment file **~templates/puppet_post_config.yaml** that registers your heat template as the **OS::TripleO::NodeExtraConfigPost:** resource type.

   ```
   resource_registry:
     OS::TripleO::NodeExtraConfigPost: /home/stack/templates/custom_puppet_config.yaml
   ```

3. Include this environment file in the **openstack overcloud deploy** command, along with any other environment files relevant to your deployment.

   ```
   $ openstack overcloud deploy --templates \
       ...
       -e /home/stack/templates/puppet_post_config.yaml \
   ```

```
...
```

This applies the configuration from **motd.pp** to all nodes in the overcloud.

CHAPTER 6. 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.

6.1. PREPARING THE UNDERCLOUD

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

Procedure

1. Log in to your undercloud as the **root** user.
2. Create the **stack** user:

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

3. Set a password for the user:

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

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

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

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

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

6. Create directories for system images and heat templates:

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

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

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

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

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

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

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

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

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

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

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

6.2. 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 17.0 entitlement:

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

5. Lock the undercloud to Red Hat Enterprise Linux 9.0:

```
$ sudo subscription-manager release --set=9.0
```

6.3. 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-9-for-x86_64-baseos-eus-rpms --enable=rhel-9-for-x86_64-appstream-eus-rpms --enable=rhel-9-for-x86_64-highavailability-eus-rpms --enable=openstack-17-for-rhel-9-x86_64-rpms --enable=fast-datapath-for-rhel-9-x86_64-rpms
```

These repositories contain packages that the director installation requires.

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

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

6.4. 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
```

6.5. 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:

```
$ 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.

6.6. 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.

Parameter	Description
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 17.0.0-5.161 and set the **modify_append_tag** to **-hotfix**, the director tags the final image as 17.0.0-5.161-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**.
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.

Parameter	Description
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 17.0.0 and **release** might be set to **5.161**, which results in the tag 17.0.0-5.161. 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.

Key	Description
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.
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.

6.7. 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 17.0.0 and the release for the container image is **5.161**, then the resulting tag for the container image is 17.0.0-5.161.

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 17.0 and 17.0.0 link to the latest **release** in the 17.0.0 container stream. If a new minor release of 17.0 occurs, the 17.0 tag links to the latest **release** for the new minor release stream while the 17.0.0 tag continues to link to the latest **release** within the 17.0.0 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 17.0. This always corresponds to the latest minor version and release.

  ```
  parameter_defaults:
    ContainerImagePrepare:
    - set:
        ...
        tag: 17.0
        ...
  ```

- To change to a specific minor version for OpenStack Platform container images, set the tag to a minor version. For example, to change to 17.0.2, set **tag** to 17.0.2.

  ```
  parameter_defaults:
    ContainerImagePrepare:
    - set:
        ...
        tag: 17.0.2
        ...
  ```

- 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: 17.0
        ...
      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": "17.0.0",
      ...
    }
  ```

- 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

17.0.0 set for **version** and 5.161 set for **release**, the resulting tag for the container image is 17.0.0-5.161.

- 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.

6.8. 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
```

6.9. 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 **17.0-hotfix**:

```
parameter_defaults:
  ContainerImagePrepare:
  - tag_from_label: "{version}-{release}"
    push_destination: true
    excludes:
    - nova-api
    set:
      namespace: registry.redhat.io/rhosp-rhel9
      name_prefix: openstack-
      name_suffix: ''
      tag:17.0
  - push_destination: true
    includes:
    - nova-api
    set:
      namespace: registry.redhat.io/rhosp-rhel9
      tag: 17.0-hotfix
```

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 match the **includes** or **excludes** regular expression value to be considered a match.

6.10. DEPLOYING A VENDOR PLUGIN

To use some third-party hardware as a Block Storage back end, you must deploy a vendor plugin. The following example demonstrates how to deploy a vendor plugin to use Dell EMC hardware as a Block Storage back end.

Procedure

1. Create a new container images file for your overcloud:

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

2. Edit the containers-prepare-parameter-dellemc.yaml file.

3. Add an **exclude** parameter to the strategy for the main Red Hat OpenStack Platform container images. Use this parameter to exclude the container image that the vendor container image will replace. In the example, the container image is the **cinder-volume** image:

   ```
   parameter_defaults:
     ContainerImagePrepare:
       - push_destination: true
         excludes:
   ```

```
      - cinder-volume
    set:
      namespace: registry.redhat.io/rhosp-rhel9
      name_prefix: openstack-
      name_suffix: ''
      tag: 16.2
      ...
    tag_from_label: "{version}-{release}"
```

4. Add a new strategy to the **ContainerImagePrepare** parameter that includes the replacement container image for the vendor plugin:

```
parameter_defaults:
  ContainerImagePrepare:
    ...
    - push_destination: true
      includes:
        - cinder-volume
      set:
        namespace: registry.connect.redhat.com/dellemc
        name_prefix: openstack-
        name_suffix: -dellemc-rhosp16
        tag: 16.2-2
        ...
```

5. Add the authentication details for the registry.connect.redhat.com registry to the **ContainerImageRegistryCredentials** parameter:

```
parameter_defaults:
  ContainerImageRegistryCredentials:
    registry.redhat.io:
      [service account username]: [service account password]
    registry.connect.redhat.com:
      [service account username]: [service account password]
```

6. Save the **containers-prepare-parameter-dellemc.yaml** file.

7. Include the **containers-prepare-parameter-dellemc.yaml** file with any deployment commands, such as as **openstack overcloud deploy**:

```
$ openstack overcloud deploy --templates
    ...
    -e containers-prepare-parameter-dellemc.yaml
    ...
```

When director deploys the overcloud, the overcloud uses the vendor container image instead of the standard container image.

IMPORTANT

The **containers-prepare-parameter-dellemc.yaml** file replaces the standard **containers-prepare-parameter.yaml** file in your overcloud deployment. Do not include the standard **containers-prepare-parameter.yaml** file in your overcloud deployment. Retain the standard **containers-prepare-parameter.yaml** file for your undercloud installation and updates.

6.11. 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.

6.12. 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.

6.13. 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
  ...
```

6.14. 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
  ...
```

6.15. 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-rhel9/openstack-nova-compute:latest

USER "root"

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

USER "nova"
```

6.16. 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-rhel9" --format="{{ .Name }}" |
sort > satellite_images
$ sudo podman search --limit 1000 "registry.redhat.io/rhceph" | grep rhceph-5-dashboard-
rhel8
$ sudo podman search --limit 1000 "registry.redhat.io/rhceph" | grep rhceph-5-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 (**OSP 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 container image:

   ```
   $ hammer repository create \
     --organization "ACME" \
     --product "OSP Containers" \
     --content-type docker \
     --url https://registry.redhat.io \
     --docker-upstream-name rhceph/rhceph-5-rhel8 \
     --upstream-username USERNAME \
     --upstream-password PASSWORD \
     --name rhceph-5-rhel8
   ```

   NOTE

   If you want to install the Ceph dashboard, include **--name rhceph-5-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-5-dashboard-rhel8 \
     --upstream-username USERNAME \
     --upstream-password PASSWORD \
     --name rhceph-5-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 of the registry on the Satellite server.

   - **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]-**

- 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-5
      ceph_namespace: satellite.example.com:5000
      ceph_tag: latest
      name_prefix: acme-production-myosp16_1-osp_containers-
      name_suffix: ''
      namespace: satellite.example.com:5000
      neutron_driver: null
      tag: '17.0'
      ...
```

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 17.0.2 image stream, set **tag: 17.0.2** 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 7. 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.

7.1. 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.

7.2. 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 only the **x86_64** architecture.

certificate_generation_ca

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

clean_nodes

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

cleanup

Cleanup temporary files. Set this to **False** to leave the temporary files used during deployment in place after you run the deployment command. 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 9.0 only supports **podman**.

container_healthcheck_disabled

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

container_images_file

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

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

container_insecure_registries

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

container_registry_mirror

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

custom_env_files

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

deployment_user

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

discovery_default_driver

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

enable_ironic; enable_ironic_inspector; enable_tempest; enable_validations

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

enable_node_discovery

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

enable_routed_networks

Defines whether to enable support for routed control plane networks.

enabled_hardware_types

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

generate_service_certificate

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

heat_container_image

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

heat_native

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

hieradata_override

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

inspection_extras

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

inspection_interface

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

inspection_runbench

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

ipv6_address_mode

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

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

ipxe_enabled

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

local_interface

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

```
2: em0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen
1000
    link/ether 52:54:00:75:24:09 brd ff:ff:ff:ff:ff:ff
    inet 192.168.122.178/24 brd 192.168.122.255 scope global dynamic em0
       valid_lft 3462sec preferred_lft 3462sec
    inet6 fe80::5054:ff:fe75:2409/64 scope link
       valid_lft forever preferred_lft forever
3: em1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noop state DOWN
    link/ether 42:0b:c2:a5:c1:26 brd ff:ff:ff:ff:ff:ff
```

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

local_ip

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

local_mtu

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

local_subnet

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

net_config_override

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

networks_file

Networks file to override for **heat**.

output_dir

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

overcloud_domain_name

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

NOTE

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

roles_file

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

scheduler_max_attempts

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

service_principal

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

subnets

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

templates

Heat templates file to override.

undercloud_admin_host

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

undercloud_debug

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

undercloud_enable_selinux

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

undercloud_hostname

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

undercloud_log_file

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

undercloud_nameservers

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

undercloud_ntp_servers

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

undercloud_public_host

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

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

undercloud_service_certificate

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

undercloud_timezone

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

undercloud_update_packages

Defines whether to update packages during the undercloud installation.

Subnets

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

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

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

IMPORTANT

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

cidr

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

masquerade

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

NOTE

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

dhcp_start; dhcp_end

The start and end of the DHCP allocation range for overcloud nodes. Ensure that this range contains enough IP addresses to allocate to your nodes. If not specified for the subnet, director determines the allocation pools by removing the values set for the **local_ip**, **gateway**, **undercloud_admin_host**, **undercloud_public_host**, and **inspection_iprange** parameters from the subnets full IP range.
You can configure non-contiguous allocation pools for undercloud control plane subnets by specifying a list of start and end address pairs. Alternatively, you can use the **dhcp_exclude** option to exclude IP addresses within an IP address range. For example, the following configurations both create allocation pools **172.20.0.100-172.20.0.150** and **172.20.0.200-172.20.0.250**:

Option 1

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

Option 2

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

dhcp_exclude

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

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

dns_nameservers

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

gateway

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

host_routes

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

inspection_iprange

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

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

7.3. 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.

7.4. 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"
```

7.5. 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
```

7.6. 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.

7.7. 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
```

7.8. 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 and configures its services according to the configuration in the **undercloud.conf**. This script takes several minutes to complete.

   The script generates two files:

   - **/home/stack/tripleo-deploy/undercloud/tripleo-undercloud-passwords.yaml** - A list of all passwords for the director services.
   - **/home/stack/stackrc** - A set of initialization variables to help you access the director command line tools.

2. The script also starts all OpenStack Platform service containers automatically. You can check the enabled containers with the following command:

   ```
   [stack@director ~]$ sudo podman ps
   ```

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.

7.9. OBTAINING IMAGES FOR OVERCLOUD NODES

Director requires several disk images to provision overcloud nodes:

- An introspection kernel and ramdisk for bare metal system introspection over PXE boot.
- A deployment kernel and ramdisk for system provisioning and deployment.
- An overcloud kernel, ramdisk, and full image, which form a base overcloud system that director writes to the hard disk of the node.

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

7.9.1. Installing the overcloud images

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

Procedure

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

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

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

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

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

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

   Skip this step if the directory already exists.

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

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

6. Import the images into director:

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

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

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

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

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

```
(undercloud) [stack@director images]$ ls -l /var/lib/ironic/httpboot
total 417296
-rwxr-xr-x. 1 root  root    6639920 Jan 29 14:48 agent.kernel
-rw-r--r--. 1 root  root  420656424 Jan 29 14:48 agent.ramdisk
-rw-r--r--. 1 42422 42422       758 Jan 29 14:29 boot.ipxe
-rw-r--r--. 1 42422 42422       488 Jan 29 14:16 inspector.ipxe
```

7.9.2. Minimal overcloud image

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

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

- **overcloud-minimal**
- **overcloud-minimal-initrd**
- **overcloud-minimal-vmlinuz**

Procedure

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

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

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

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

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

```
(undercloud) [stack@director ~]$ cd ~/images
(undercloud) [stack@director images]$ tar xf /usr/share/rhosp-director-images/overcloud-minimal-latest-17.0.tar
```

5. Import the images into director:

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

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

```
Image "file:///var/lib/ironic/images/overcloud-minimal.vmlinuz" was copied.
+--------------------------------------------------------+-------------------+----------+
|                          Path                          |        Name       |   Size   |
+--------------------------------------------------------+-------------------+----------+
| file:///var/lib/ironic/images/overcloud-minimal.vmlinuz | overcloud-minimal | 11172880 |
+--------------------------------------------------------+-------------------+----------+
Image "file:///var/lib/ironic/images/overcloud-minimal.initrd" was copied.
+------------------------------------------------------+-------------------+----------+
|                         Path                         |        Name       |   Size   |
+------------------------------------------------------+-------------------+----------+
| file:///var/lib/ironic/images/overcloud-minimal.initrd | overcloud-minimal | 63575845 |
+------------------------------------------------------+-------------------+----------+
Image "file:///var/lib/ironic/images/overcloud-minimal.raw" was copied.
+---------------------------------------------------+-------------------+------------+
|                         Path                      |        Name       |    Size    |
+---------------------------------------------------+-------------------+------------+
| file:///var/lib/ironic/images/overcloud-minimal.raw | overcloud-minimal | 2912878592 |
+---------------------------------------------------+-------------------+------------+
```

7.10. 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.

7.11. UNDERCLOUD CONTAINER REGISTRY

Red Hat Enterprise Linux 9.0 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 8. 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.

8.1. 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 8.1. 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 *Director installation and usage* guide.

Table 8.2. 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.

8.2. 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.

8.3. OVERCLOUD STORAGE

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

Images

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

Volumes

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

Objects

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

File Systems

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

Instance disks

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

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

8.3.1. Configuration considerations for overcloud storage nodes

Instance security and performance

Using LVM on an instance that uses a back end Block Storage volume causes issues with performance, volume visibility and availability, and data corruption. Use an LVM filter to mitigate visibility, availability, and data corruption issues. For more information, see Enabling LVM2 filtering on overcloud nodes in the *Storage Guide*, and the Red Hat Knowledgebase solution Using LVM on a cinder volume exposes the data to the compute host.

Local disk partition sizes

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

Partition	Default size
/	8GB
/tmp	1GB
/var/log	10GB
/var/log/audit	2GB
/home	1GB
/var	Allocated the remaining size of the disk after all other partitions are allocated.

To change the allocated disk size for a partition, update the **role_growvols_args** extra Ansible variable in the Ansible_playbooks definition in your **overcloud-baremetal-deploy.yaml** node definition file. For more information, see Provisioning bare metal nodes for the overcloud .

If your partitions continue to fill up after you have optimized the configuration of your partition sizes, then perform one of the following tasks:

- Manually delete files from the affected partitions.
- Add a new physical disk and add it to the LVM volume group. For more information, see Configuring and managing logical volumes.

NOTE

Adding a new disk requires a support exception. Contact the Red Hat Customer Experience and Engagement team to discuss a support exception, if applicable, or other options.

8.4. OVERCLOUD SECURITY

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

- Use network segmentation to mitigate network movement and isolate sensitive data. A flat network is much less secure.
- Restrict services access and ports to a minimum.
- Enforce proper firewall rules and password usage.

- Ensure that SELinux is enabled.

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

- Security Hardening for Red Hat Enterprise Linux 9
- Using SELinux for Red Hat Enterprise Linux 9

8.5. 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.

8.6. CONTROLLER NODE REQUIREMENTS

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

Processor

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

Memory

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

- **Controller RAM minimum calculation:**
  - Use 1.5 GB of memory for each vCPU. For example, a machine with 48 vCPUs should have 72 GB of RAM.
- **Controller RAM recommended calculation:**
  - Use 3 GB of memory for each vCPU. For example, a machine with 48 vCPUs should have 144 GB of RAM

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

Disk Storage and layout

A minimum amount of 50 GB storage is required if the Object Storage service (swift) is not running on the Controller nodes. However, the Telemetry and Object Storage services are both installed on the Controllers, with both configured to use the root disk. These defaults are suitable for deploying small overclouds built on commodity hardware. These environments are typical of proof-of-concept and test environments. You can use these defaults to deploy overclouds with minimal planning, but they offer little in terms of workload capacity and performance.
In an enterprise environment, however, the defaults could cause a significant bottleneck because Telemetry accesses storage constantly. This results in heavy disk I/O usage, which severely impacts the performance of all other Controller services. In this type of environment, you must plan your overcloud and configure it accordingly.

Network Interface Cards

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

Power management

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

Virtualization support

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

8.7. COMPUTE NODE REQUIREMENTS

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

Processor

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

Memory

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

- Add additional memory that you intend to make available to virtual machine instances.
- Add additional memory to run special features or additional resources on the host, such as additional kernel modules, virtual switches, monitoring solutions, and other additional background tasks.
- If you intend to use non-uniform memory access (NUMA), Red Hat recommends 8GB per CPU socket node or 16 GB per socket node if you have more then 256 GB of physical RAM.
- Configure at least 4 GB of swap space.

Disk space

A minimum of 50 GB of available disk space.

Network Interface Cards

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

Power management

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

8.8. RED HAT CEPH STORAGE NODE REQUIREMENTS

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

- Hardware requirements including processor, memory, and network interface card selection and disk layout are available in the Red Hat Ceph Storage Hardware Guide .
- Each Ceph Storage node requires a supported power management interface, such as Intelligent Platform Management Interface (IPMI) functionality, on the motherboard of the server.
- Each Ceph Storage node must have at least two disks. RHOSP director uses **cephadm** to deploy the Ceph Storage cluster. The cephadm functionality does not support installing Ceph OSD on the root disk of the node.

8.9. CEPH STORAGE NODES AND RHEL COMPATIBILITY

RHOSP 17.0 is supported on RHEL 9.0. However, hosts that are mapped to the Ceph Storage role update to the latest major RHEL release. Before upgrading, review the Red Hat Knowledgebase article Red Hat Ceph Storage: Supported configurations .

8.10. 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.

8.11. OVERCLOUD REPOSITORIES

You run Red Hat OpenStack Platform 17.0 on Red Hat Enterprise Linux 9.0. As a result, you must lock the content from these repositories to the respective Red Hat Enterprise Linux version.

WARNING

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

NOTE

Satellite repositories are not listed because RHOSP 17.0 does not support Satellite. Satellite support is planned for a future release. Only Red Hat CDN is supported as a package repository and container registry. NFV repositories are not listed because RHOSP 17.0 does not support NFV.

Controller node repositories

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

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

Name	Repository	Description of requirement
Red Hat Enterprise Linux 9 for x86_64 – High Availability (RPMs) Extended Update Support (EUS)	**rhel-9-for-x86_64-highavailability-eus-rpms**	High availability tools for Red Hat Enterprise Linux.
Red Hat OpenStack Platform 17 for RHEL 9 x86_64 (RPMs)	**openstack-17-for-rhel-9-x86_64-rpms**	Core Red Hat OpenStack Platform repository.
Red Hat Fast Datapath for RHEL 9 (RPMS)	**fast-datapath-for-rhel-9-x86_64-rpms**	Provides Open vSwitch (OVS) packages for OpenStack Platform.
Red Hat Ceph Storage Tools 5 for RHEL 9 x86_64 (RPMs)	**rhceph-5-tools-for-rhel-9-x86_64-rpms**	Tools for Red Hat Ceph Storage 5 for Red Hat Enterprise Linux 9.

Compute and ComputeHCI node repositories

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

Name	Repository	Description of requirement
Red Hat Enterprise Linux 9 for x86_64 – BaseOS (RPMs) Extended Update Support (EUS)	**rhel-9-for-x86_64-baseos-eus-rpms**	Base operating system repository for x86_64 systems.
Red Hat Enterprise Linux 9 for x86_64 – AppStream (RPMs)	**rhel-9-for-x86_64-appstream-eus-rpms**	Contains Red Hat OpenStack Platform dependencies.
Red Hat Enterprise Linux 9 for x86_64 – High Availability (RPMs) Extended Update Support (EUS)	**rhel-9-for-x86_64-highavailability-eus-rpms**	High availability tools for Red Hat Enterprise Linux.
Red Hat OpenStack Platform 17 for RHEL 9 x86_64 (RPMs)	**openstack-17-for-rhel-9-x86_64-rpms**	Core Red Hat OpenStack Platform repository.
Red Hat Fast Datapath for RHEL 9 (RPMS)	**fast-datapath-for-rhel-9-x86_64-rpms**	Provides Open vSwitch (OVS) packages for OpenStack Platform.
Red Hat Ceph Storage Tools 5 for RHEL 9 x86_64 (RPMs)	**rhceph-5-tools-for-rhel-9-x86_64-rpms**	Tools for Red Hat Ceph Storage 5 for Red Hat Enterprise Linux 9.

Real Time Compute repositories

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

Name	Repository	Description of requirement
Red Hat Enterprise Linux 9 for x86_64 – Real Time (RPMs)	**rhel-9-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.

Ceph Storage node repositories

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

Name	Repository	Description of requirement
Red Hat Enterprise Linux 9 for x86_64 – BaseOS (RPMs)	**rhel-9-for-x86_64-baseos-rpms**	Base operating system repository for x86_64 systems.
Red Hat Enterprise Linux 9 for x86_64 – AppStream (RPMs)	**rhel-9-for-x86_64-appstream-rpms**	Contains Red Hat OpenStack Platform dependencies.
Red Hat OpenStack Platform 17 Director Deployment Tools for RHEL 9 x86_64 (RPMs)	**openstack-17-deployment-tools-for-rhel-9-x86_64-rpms**	Packages to help director configure Ceph Storage nodes. This repository is included with standalone Ceph Storage subscriptions. If you use a combined OpenStack Platform and Ceph Storage subscription, use the **openstack-17-for-rhel-9-x86_64-rpms** repository.
Red Hat OpenStack Platform 17 for RHEL 9 x86_64 (RPMs)	**openstack-17-for-rhel-9-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-17-deployment-tools-for-rhel-9-x86_64-rpms** repository.
Red Hat Ceph Storage Tools 5 for RHEL 9 x86_64 (RPMs)	**rhceph-5-tools-for-rhel-9-x86_64-rpms**	Provides tools for nodes to communicate with the Ceph Storage cluster.

Name	Repository	Description of requirement
Red Hat Fast Datapath for RHEL 9 (RPMS)	**fast-datapath-for-rhel-9-x86_64-rpms**	Provides Open vSwitch (OVS) packages for OpenStack Platform. If you are using OVS on Ceph Storage nodes, add this repository to the network interface configuration (NIC) templates.

8.12. NODE PROVISIONING AND CONFIGURATION

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

Provisioning with the OpenStack Bare Metal (ironic) service

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

Provisioning with an external tool

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

CHAPTER 9. COMPOSABLE SERVICES AND CUSTOM ROLES

The overcloud usually consists of nodes in predefined roles such as Controller nodes, Compute nodes, and different storage node types. Each of these default roles contains a set of services defined in the core heat template collection on the director node. However, you can also create custom roles that contain specific sets of services.

You can use this flexibility to create different combinations of services on different roles. This chapter explores the architecture of custom roles, composable services, and methods for using them.

9.1. SUPPORTED ROLE ARCHITECTURE

The following architectures are available when you use custom roles and composable services:

Default architecture

Uses the default **roles_data** files. All controller services are contained within one Controller role.

Supported standalone roles

Use the predefined files in **/usr/share/openstack-tripleo-heat-templates/roles** to generate a custom **roles_data** file. For more information, see Section 9.4, “Supported custom roles” .

Custom composable services

Create your own roles and use them to generate a custom **roles_data** file. Note that only a limited number of composable service combinations have been tested and verified and Red Hat cannot support all composable service combinations.

9.2. EXAMINING THE ROLES_DATA FILE

The **roles_data** file contains a YAML-formatted list of the roles that director deploys onto nodes. Each role contains definitions of all of the services that comprise the role. Use the following example snippet to understand the **roles_data** syntax:

```
- name: Controller
  description: |
    Controller role that has all the controller services loaded and handles
    Database, Messaging and Network functions.
  ServicesDefault:
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CephClient
    ...
- name: Compute
  description: |
    Basic Compute Node role
  ServicesDefault:
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CephClient
    ...
```

The core heat template collection contains a default **roles_data** file located at **/usr/share/openstack-tripleo-heat-templates/roles_data.yaml**. The default file contains definitions of the following role types:

- **Controller**

- **Compute**
- **BlockStorage**
- **ObjectStorage**
- **CephStorage**.

The **openstack overcloud deploy** command includes the default **roles_data.yaml** file during deployment. However, you can use the **-r** argument to override this file with a custom **roles_data** file:

```
$ openstack overcloud deploy --templates -r ~/templates/roles_data-custom.yaml
```

9.3. CREATING A ROLES_DATA FILE

Although you can create a custom **roles_data** file manually, you can also generate the file automatically using individual role templates. Director provides the **openstack overcloud role generate** command to join multiple predefined roles and automatically generate a custom **roles_data** file.

Procedure

1. List the default role templates:

   ```
   $ openstack overcloud role list
   BlockStorage
   CephStorage
   Compute
   ComputeHCI
   ComputeOvsDpdk
   Controller
   ...
   ```

2. View the role definition:

   ```
   $ openstack overcloud role show Compute
   ```

3. Generate a custom **roles_data.yaml** file that contains the **Controller**, **Compute**, and **Networker** roles:

   ```
   $ openstack overcloud roles \
    generate -o <custom_role_file> \
    Controller Compute Networker
   ```

   - Replace **<custom_role_file>** with the name and location of the new role file to generate, for example, **/home/stack/templates/roles_data.yaml**.

   NOTE

   The **Controller** and **Networker** roles contain the same networking agents. This means that the networking services scale from the **Controller** role to the **Networker** role and the overcloud balances the load for networking services between the **Controller** and **Networker** nodes.

To make this **Networker** role standalone, you can create your own custom **Controller** role, as well as any other role that you require. This allows you to generate a **roles_data.yaml** file from your own custom roles.

4. Copy the **roles** directory from the core heat template collection to the home directory of the **stack** user:

```
$ cp -r /usr/share/openstack-tripleo-heat-templates/roles/. /home/stack/templates/roles/
```

5. Add or modify the custom role files in this directory. Use the **--roles-path** option with any of the role sub-commands to use this directory as the source for your custom roles:

```
$ openstack overcloud role \
 generate -o my_roles_data.yaml \
 --roles-path /home/stack/templates/roles \
 Controller Compute Networker
```

This command generates a single **my_roles_data.yaml** file from the individual roles in the **~/roles** directory.

NOTE

The default roles collection also contains the **ControllerOpenstack** role, which does not include services for **Networker**, **Messaging**, and **Database** roles. You can use the **ControllerOpenstack** in combination with the standalone **Networker**, **Messaging**, and **Database** roles.

9.4. SUPPORTED CUSTOM ROLES

The following table contains information about the available custom roles. You can find custom role templates in the **/usr/share/openstack-tripleo-heat-templates/roles** directory.

Role	Description	File
BlockStorage	OpenStack Block Storage (cinder) node.	**BlockStorage.yaml**
CephAll	Full standalone Ceph Storage node. Includes OSD, MON, Object Gateway (RGW), Object Operations (MDS), Manager (MGR), and RBD Mirroring.	**CephAll.yaml**
CephFile	Standalone scale-out Ceph Storage file role. Includes OSD and Object Operations (MDS).	**CephFile.yaml**
CephObject	Standalone scale-out Ceph Storage object role. Includes OSD and Object Gateway (RGW).	**CephObject.yaml**
CephStorage	Ceph Storage OSD node role.	**CephStorage.yaml**
ComputeAlt	Alternate Compute node role.	**ComputeAlt.yaml**
ComputeDVR	DVR enabled Compute node role.	**ComputeDVR.yaml**

Role	Description	File
ComputeHCI	Compute node with hyper-converged infrastructure. Includes Compute and Ceph OSD services.	**ComputeHCI.yaml**
ComputeInstanceHA	Compute Instance HA node role. Use in conjunction with the **environments/compute-instanceha.yaml`** environment file.	**ComputeInstanceHA.yaml**
ComputeLiquidio	Compute node with Cavium Liquidio Smart NIC.	**ComputeLiquidio.yaml**
ComputeOvsDpdkRT	Compute OVS DPDK RealTime role.	**ComputeOvsDpdkRT.yaml**
ComputeOvsDpdk	Compute OVS DPDK role.	**ComputeOvsDpdk.yaml**
ComputeRealTime	Compute role optimized for real-time behaviour. When using this role, it is mandatory that an **overcloud-realtime-compute** image is available and the role specific parameters **IsolCpusList**, **NovaComputeCpuDedicatedSet** and **NovaComputeCpuSharedSet** are set according to the hardware of the real-time compute nodes.	**ComputeRealTime.yaml**
ComputeSriovRT	Compute SR-IOV RealTime role.	**ComputeSriovRT.yaml**
ComputeSriov	Compute SR-IOV role.	**ComputeSriov.yaml**
Compute	Standard Compute node role.	**Compute.yaml**
ControllerAllNovaStandalone	Controller role that does not contain the database, messaging, networking, and OpenStack Compute (nova) control components. Use in combination with the **Database**, **Messaging**, **Networker**, and **Novacontrol** roles.	**ControllerAllNovaStandalone.yaml**
ControllerNoCeph	Controller role with core Controller services loaded but no Ceph Storage (MON) components. This role handles database, messaging, and network functions but not any Ceph Storage functions.	**ControllerNoCeph.yaml**
ControllerNovaStandalone	Controller role that does not contain the OpenStack Compute (nova) control component. Use in combination with the **Novacontrol** role.	**ControllerNovaStandalone.yaml**

Role	Description	File
ControllerOpenstack	Controller role that does not contain the database, messaging, and networking components. Use in combination with the **Database**, **Messaging**, and **Networker** roles.	**ControllerOpenstack.yaml**
ControllerStorageNfs	Controller role with all core services loaded and uses Ceph NFS. This roles handles database, messaging, and network functions.	**ControllerStorageNfs.yaml**
Controller	Controller role with all core services loaded. This roles handles database, messaging, and network functions.	**Controller.yaml**
ControllerSriov (ML2/OVN)	Same as the normal Controller role but with the OVN Metadata agent deployed.	**ControllerSriov.yaml**
Database	Standalone database role. Database managed as a Galera cluster using Pacemaker.	**Database.yaml**
HciCephAll	Compute node with hyper-converged infrastructure and all Ceph Storage services. Includes OSD, MON, Object Gateway (RGW), Object Operations (MDS), Manager (MGR), and RBD Mirroring.	**HciCephAll.yaml**
HciCephFile	Compute node with hyper-converged infrastructure and Ceph Storage file services. Includes OSD and Object Operations (MDS).	**HciCephFile.yaml**
HciCephMon	Compute node with hyper-converged infrastructure and Ceph Storage block services. Includes OSD, MON, and Manager.	**HciCephMon.yaml**
HciCephObject	Compute node with hyper-converged infrastructure and Ceph Storage object services. Includes OSD and Object Gateway (RGW).	**HciCephObject.yaml**
IronicConductor	Ironic Conductor node role.	**IronicConductor.yaml**
Messaging	Standalone messaging role. RabbitMQ managed with Pacemaker.	**Messaging.yaml**
Networker	Standalone networking role. Runs OpenStack networking (neutron) agents on their own. If your deployment uses the ML2/OVN mechanism driver, see additional steps in Deploying a Custom Role with ML2/OVN.	**Networker.yaml**

Role	Description	File
NetworkerSriov	Same as the normal Networker role but with the OVN Metadata agent deployed. See additional steps in Deploying a Custom Role with ML2/OVN	**NetworkerSriov.yaml**
Novacontrol	Standalone **nova-control** role to run OpenStack Compute (nova) control agents on their own.	**Novacontrol.yaml**
ObjectStorage	Swift Object Storage node role.	**ObjectStorage.yaml**
Telemetry	Telemetry role with all the metrics and alarming services.	**Telemetry.yaml**

9.5. EXAMINING ROLE PARAMETERS

Each role contains the following parameters:

name

(Mandatory) The name of the role, which is a plain text name with no spaces or special characters. Check that the chosen name does not cause conflicts with other resources. For example, use **Networker** as a name instead of **Network**.

description

(Optional) A plain text description for the role.

tags

(Optional) A YAML list of tags that define role properties. Use this parameter to define the primary role with both the **controller** and **primary** tags together:

```
- name: Controller
  ...
  tags:
    - primary
    - controller
  ...
```

IMPORTANT

If you do not tag the primary role, the first role that you define becomes the primary role. Ensure that this role is the Controller role.

networks

A YAML list or dictionary of networks that you want to configure on the role. If you use a YAML list, list each composable network:

```
  networks:
    - External
    - InternalApi
    - Storage
    - StorageMgmt
    - Tenant
```

If you use a dictionary, map each network to a specific **subnet** in your composable networks.

```
  networks:
    External:
      subnet: external_subnet
    InternalApi:
      subnet: internal_api_subnet
    Storage:
      subnet: storage_subnet
    StorageMgmt:
      subnet: storage_mgmt_subnet
    Tenant:
      subnet: tenant_subnet
```

Default networks include **External**, **InternalApi**, **Storage**, **StorageMgmt**, **Tenant**, and **Management**.

CountDefault

(Optional) Defines the default number of nodes that you want to deploy for this role.

HostnameFormatDefault

(Optional) Defines the default hostname format for the role. The default naming convention uses the following format:

```
[STACK NAME]-[ROLE NAME]-[NODE ID]
```

For example, the default Controller nodes are named:

```
overcloud-controller-0
overcloud-controller-1
overcloud-controller-2
...
```

disable_constraints

(Optional) Defines whether to disable OpenStack Compute (nova) and OpenStack Image Storage (glance) constraints when deploying with director. Use this parameter when you deploy an overcloud with pre-provisioned nodes. For more information, see Configuring a basic overcloud with pre-provisioned nodes in the *Director installation and usage* guide.

update_serial

(Optional) Defines how many nodes to update simultaneously during the OpenStack update options. In the default **roles_data.yaml** file:

- The default is **1** for Controller, Object Storage, and Ceph Storage nodes.
- The default is **25** for Compute and Block Storage nodes.

If you omit this parameter from a custom role, the default is **1**.

ServicesDefault

(Optional) Defines the default list of services to include on the node. For more information, see Section 9.8, “Examining composable service architecture” .

You can use these parameters to create new roles and also define which services to include in your roles.

The **openstack overcloud deploy** command integrates the parameters from the **roles_data** file into some of the Jinja2-based templates. For example, at certain points, the **overcloud.j2.yaml** heat template iterates over the list of roles from **roles_data.yaml** and creates parameters and resources specific to each respective role.

For example, the following snippet contains the resource definition for each role in the **overcloud.j2.yaml** heat template:

```
  {{role.name}}:
    type: OS::Heat::ResourceGroup
    depends_on: Networks
    properties:
      count: {get_param: {{role.name}}Count}
      removal_policies: {get_param: {{role.name}}RemovalPolicies}
      resource_def:
        type: OS::TripleO::{{role.name}}
        properties:
          CloudDomain: {get_param: CloudDomain}
          ServiceNetMap: {get_attr: [ServiceNetMap, service_net_map]}
          EndpointMap: {get_attr: [EndpointMap, endpoint_map]}
...
```

This snippet shows how the Jinja2-based template incorporates the **{{role.name}}** variable to define the name of each role as an **OS::Heat::ResourceGroup** resource. This in turn uses each **name** parameter from the **roles_data** file to name each respective **OS::Heat::ResourceGroup** resource.

9.6. CREATING A NEW ROLE

You can use the composable service architecture to assign roles to bare-metal nodes according to the requirements of your deployment. For example, you might want to create a new **Horizon** role to host only the OpenStack Dashboard (**horizon**).

Procedure

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

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

3. Copy the **roles** directory from the core heat template collection to the home directory of the **stack** user:

   ```
   $ cp -r /usr/share/openstack-tripleo-heat-templates/roles/. /home/stack/templates/roles/
   ```

4. Create a new file named **Horizon.yaml** in **home/stack/templates/roles**.
5. Add the following configuration to **Horizon.yaml** to create a new **Horizon** role that contains the base and core OpenStack Dashboard services:

```
- name: Horizon (1)
  CountDefault: 1 (2)
  HostnameFormatDefault: '%stackname%-horizon-%index%'
  ServicesDefault:
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::Ntp
    - OS::TripleO::Services::Snmp
    - OS::TripleO::Services::Sshd
    - OS::TripleO::Services::Timezone
    - OS::TripleO::Services::TripleoPackages
    - OS::TripleO::Services::TripleoFirewall
    - OS::TripleO::Services::SensuClient
    - OS::TripleO::Services::FluentdClient
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::Collectd
    - OS::TripleO::Services::MySQLClient
    - OS::TripleO::Services::Apache
    - OS::TripleO::Services::Horizon
```

(1) Set the **name** parameter to the name of the custom role. Custom role names have a maximum length of 47 characters.

(2) Set the **CountDefault** parameter to **1** so that a default overcloud always includes the **Horizon** node.

6. Optional: If you want to scale the services in an existing overcloud, retain the existing services on the **Controller** role. If you want to create a new overcloud and you want the OpenStack Dashboard to remain on the standalone role, remove the OpenStack Dashboard components from the **Controller** role definition:

```
- name: Controller
  CountDefault: 1
  ServicesDefault:
    ...
    - OS::TripleO::Services::GnocchiMetricd
    - OS::TripleO::Services::GnocchiStatsd
    - OS::TripleO::Services::HAproxy
    - OS::TripleO::Services::HeatApi
    - OS::TripleO::Services::HeatApiCfn
    - OS::TripleO::Services::HeatApiCloudwatch
    - OS::TripleO::Services::HeatEngine
    # - OS::TripleO::Services::Horizon          # Remove this service
    - OS::TripleO::Services::IronicApi
    - OS::TripleO::Services::IronicConductor
    - OS::TripleO::Services::Iscsid
    - OS::TripleO::Services::Keepalived
    ...
```

7. Generate a new roles data file named **roles_data_horizon.yaml** that includes the **Controller**, **Compute**, and **Horizon** roles:

```
(undercloud)$ openstack overcloud roles \
 generate -o /home/stack/templates/roles_data_horizon.yaml \
--roles-path /home/stack/templates/roles \
Controller Compute Horizon
```

8. Optional: Edit the **overcloud-baremetal-deploy.yaml** node definition file to configure the placement of the Horizon node:

```
- name: Controller
  count: 3
  instances:
  - hostname: overcloud-controller-0
    name: node00
  ...
- name: Compute
  count: 3
  instances:
  - hostname: overcloud-novacompute-0
    name: node04
  ...
- name: Horizon
  count: 1
  instances:
  - hostname: overcloud-horizon-0
    name: node07
```

9.7. GUIDELINES AND LIMITATIONS

Note the following guidelines and limitations for the composable role architecture.

For services not managed by Pacemaker:

- You can assign services to standalone custom roles.
- You can create additional custom roles after the initial deployment and deploy them to scale existing services.

For services managed by Pacemaker:

- You can assign Pacemaker-managed services to standalone custom roles.
- Pacemaker has a 16 node limit. If you assign the Pacemaker service (**OS::TripleO::Services::Pacemaker**) to 16 nodes, subsequent nodes must use the Pacemaker Remote service (**OS::TripleO::Services::PacemakerRemote**) instead. You cannot have the Pacemaker service and Pacemaker Remote service on the same role.
- Do not include the Pacemaker service (**OS::TripleO::Services::Pacemaker**) on roles that do not contain Pacemaker-managed services.
- You cannot scale up or scale down a custom role that contains **OS::TripleO::Services::Pacemaker** or **OS::TripleO::Services::PacemakerRemote** services.

General limitations:

- You cannot change custom roles and composable services during a major version upgrade.

- You cannot modify the list of services for any role after deploying an overcloud. Modifying the service lists after Overcloud deployment can cause deployment errors and leave orphaned services on nodes.

9.8. EXAMINING COMPOSABLE SERVICE ARCHITECTURE

The core heat template collection contains two sets of composable service templates:

- **deployment** contains the templates for key OpenStack services.
- **puppet/services** contains legacy templates for configuring composable services. In some cases, the composable services use templates from this directory for compatibility. In most cases, the composable services use the templates in the **deployment** directory.

Each template contains a description that identifies its purpose. For example, the **deployment/time/ntp-baremetal-puppet.yaml** service template contains the following description:

```
description: >
  NTP service deployment using puppet, this YAML file
  creates the interface between the HOT template
  and the puppet manifest that actually installs
  and configure NTP.
```

These service templates are registered as resources specific to a Red Hat OpenStack Platform deployment. This means that you can call each resource using a unique heat resource namespace defined in the **overcloud-resource-registry-puppet.j2.yaml** file. All services use the **OS::TripleO::Services** namespace for their resource type.

Some resources use the base composable service templates directly:

```
resource_registry:
  ...
  OS::TripleO::Services::Ntp: deployment/time/ntp-baremetal-puppet.yaml
  ...
```

However, core services require containers and use the containerized service templates. For example, the **keystone** containerized service uses the following resource:

```
resource_registry:
  ...
  OS::TripleO::Services::Keystone: deployment/keystone/keystone-container-puppet.yaml
  ...
```

These containerized templates usually reference other templates to include dependencies. For example, the **deployment/keystone/keystone-container-puppet.yaml** template stores the output of the base template in the **ContainersCommon** resource:

```
resources:
  ContainersCommon:
    type: ../containers-common.yaml
```

The containerized template can then incorporate functions and data from the **containers-common.yaml** template.

The **overcloud.j2.yaml** heat template includes a section of Jinja2-based code to define a service list for each custom role in the **roles_data.yaml** file:

```
{{role.name}}Services:
  description: A list of service resources (configured in the heat
               resource_registry) which represent nested stacks
               for each service that should get installed on the {{role.name}} role.
  type: comma_delimited_list
  default: {{role.ServicesDefault|default([])}}
```

For the default roles, this creates the following service list parameters: **ControllerServices**, **ComputeServices**, **BlockStorageServices**, **ObjectStorageServices**, and **CephStorageServices**.

You define the default services for each custom role in the **roles_data.yaml** file. For example, the default Controller role contains the following content:

```
- name: Controller
  CountDefault: 1
  ServicesDefault:
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CephMon
    - OS::TripleO::Services::CephExternal
    - OS::TripleO::Services::CephRgw
    - OS::TripleO::Services::CinderApi
    - OS::TripleO::Services::CinderBackup
    - OS::TripleO::Services::CinderScheduler
    - OS::TripleO::Services::CinderVolume
    - OS::TripleO::Services::Core
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::Keystone
    - OS::TripleO::Services::GlanceApi
    - OS::TripleO::Services::GlanceRegistry
...
```

These services are then defined as the default list for the **ControllerServices** parameter.

NOTE

You can also use an environment file to override the default list for the service parameters. For example, you can define **ControllerServices** as a **parameter_default** in an environment file to override the services list from the **roles_data.yaml** file.

9.9. ADDING AND REMOVING SERVICES FROM ROLES

The basic method of adding or removing services involves creating a copy of the default service list for a node role and then adding or removing services. For example, you might want to remove OpenStack Orchestration (heat) from the Controller nodes.

Procedure

1. Create a custom copy of the default **roles** directory:

   ```
   $ cp -r /usr/share/openstack-tripleo-heat-templates/roles ~/.
   ```

2. Edit the **~/roles/Controller.yaml** file and modify the service list for the **ServicesDefault** parameter. Scroll to the OpenStack Orchestration services and remove them:

```
    - OS::TripleO::Services::GlanceApi
    - OS::TripleO::Services::GlanceRegistry
    - OS::TripleO::Services::HeatApi           # Remove this service
    - OS::TripleO::Services::HeatApiCfn        # Remove this service
    - OS::TripleO::Services::HeatApiCloudwatch  # Remove this service
    - OS::TripleO::Services::HeatEngine        # Remove this service
    - OS::TripleO::Services::MySQL
    - OS::TripleO::Services::NeutronDhcpAgent
```

3. Generate the new **roles_data** file:

```
$ openstack overcloud roles generate -o roles_data-no_heat.yaml \
  --roles-path ~/roles \
  Controller Compute Networker
```

4. Include this new **roles_data** file when you run the **openstack overcloud deploy** command:

```
$ openstack overcloud deploy --templates -r ~/templates/roles_data-no_heat.yaml
```

This command deploys an overcloud without OpenStack Orchestration services installed on the Controller nodes.

NOTE

You can also disable services in the **roles_data** file using a custom environment file. Redirect the services to disable to the **OS::Heat::None** resource. For example:

```
resource_registry:
  OS::TripleO::Services::HeatApi: OS::Heat::None
  OS::TripleO::Services::HeatApiCfn: OS::Heat::None
  OS::TripleO::Services::HeatApiCloudwatch: OS::Heat::None
  OS::TripleO::Services::HeatEngine: OS::Heat::None
```

9.10. ENABLING DISABLED SERVICES

Some services are disabled by default. These services are registered as null operations (**OS::Heat::None**) in the **overcloud-resource-registry-puppet.j2.yaml** file. For example, the Block Storage backup service (**cinder-backup**) is disabled:

```
OS::TripleO::Services::CinderBackup: OS::Heat::None
```

To enable this service, include an environment file that links the resource to its respective heat templates in the **puppet/services** directory. Some services have predefined environment files in the **environments** directory. For example, the Block Storage backup service uses the **environments/cinder-backup.yaml** file, which contains the following entry:

Procedure

1. Add an entry in an environment file that links the **CinderBackup** service to the heat template that contains the **cinder-backup** configuration:

```
resource_registry:
  OS::TripleO::Services::CinderBackup: ../podman/services/pacemaker/cinder-backup.yaml
...
```

This entry overrides the default null operation resource and enables the service.

2. Include this environment file when you run the **openstack overcloud deploy** command:

```
$ openstack overcloud deploy --templates -e /usr/share/openstack-tripleo-heat-
templates/environments/cinder-backup.yaml
```

9.11. CREATING A GENERIC NODE WITH NO SERVICES

You can create generic Red Hat Enterprise Linux 9.0 nodes without any OpenStack services configured. This is useful when you need to host software outside of the core Red Hat OpenStack Platform (RHOSP) environment. For example, RHOSP provides integration with monitoring tools such as Kibana and Sensu. For more information, see the *Monitoring Tools Configuration Guide* . While Red Hat does not provide support for the monitoring tools themselves, director can create a generic Red Hat Enterprise Linux 9.0 node to host these tools.

NOTE

The generic node still uses the base **overcloud-hardened-uefi-full.qcow2** image rather than a base Red Hat Enterprise Linux 9 image. This means the node has some Red Hat OpenStack Platform software installed but not enabled or configured.

Procedure

1. Create a generic role in your custom **roles_data.yaml** file that does not contain a **ServicesDefault** list:

```
- name: Generic
- name: Controller
  CountDefault: 1
  ServicesDefault:
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CephClient
    ...
- name: Compute
  CountDefault: 1
  ServicesDefault:
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CephClient
    ...
```

Ensure that you retain the existing **Controller** and **Compute** roles.

2. Create an environment file **generic-node-params.yaml** to specify how many generic Red Hat Enterprise Linux 9 nodes you require and the flavor when selecting nodes to provision:

```
parameter_defaults:
  OvercloudGenericFlavor: baremetal
  GenericCount: 1
```

3. Include both the roles file and the environment file when you run the **openstack overcloud deploy** command:

```
$ openstack overcloud deploy --templates \
-r ~/templates/roles_data_with_generic.yaml \
-e ~/templates/generic-node-params.yaml
```

This configuration deploys a three-node environment with one Controller node, one Compute node, and one generic Red Hat Enterprise Linux 9 node.

CHAPTER 10. CONFIGURING OVERCLOUD NETWORKING

To configure the physical network for your overcloud, create the following configuration files:

- The network configuration file, **network_data.yaml**, that follows the structure defined in the network data schema.
- The network interface controllers (NICs) configuration files, by using the NIC template file in Jinja2 ansible format, **j2**.

10.1. EXAMPLE NETWORK CONFIGURATION FILES

The following are examples of the network data schema for IPv4 and IPv6.

10.1.1. Example network data schema for IPv4

```
- name: Storage
  name_lower: storage                    #optional, default: name.lower
  admin_state_up: false                  #optional, default: false
  dns_domain: storage.localdomain.       #optional, default: undef
  mtu: 1442                              #optional, default: 1500
  shared: false                          #optional, default: false
  service_net_map_replace: storage       #optional, default: undef
  ipv6: false                            #optional, default: false
  vip: true                              #optional, default: false
  subnets:
    subnet01:
      ip_subnet: 172.18.1.0/24
      gateway_ip: 172.18.1.254           #optional, default: undef
      allocation_pools:                  #optional, default: []
        - start: 172.18.1.10
          end: 172.18.1.250
      enable_dhcp: false                 #optional, default: false
      routes:                            #optional, default: []
        - destination: 172.18.0.0/24
          nexthop: 172.18.1.254
      vlan: 21                           #optional, default: undef
      physical_network: storage_subnet01  #optional, default: {{name.lower}}_{{subnet name}}
      network_type: flat                 #optional, default: flat
      segmentation_id: 21                #optional, default: undef
    subnet02:
      ip_subnet: 172.18.0.0/24
      gateway_ip: 172.18.0.254           #optional, default: undef
      allocation_pools:                  #optional, default: []
        - start: 172.18.0.10
          end: 172.18.0.250
      enable_dhcp: false                 #optional, default: false
      routes:                            #optional, default: []
        - destination: 172.18.1.0/24
          nexthop: 172.18.0.254
      vlan: 20                           #optional, default: undef
      physical_network: storage_subnet02  #optional, default: {{name.lower}}_{{subnet name}}
      network_type: flat                 #optional, default: flat
      segmentation_id: 20                #optional, default: undef
```

10.1.2. Example network data schema for IPv6

```
- name: Storage
  name_lower: storage
  admin_state_up: false
  dns_domain: storage.localdomain.
  mtu: 1442
  shared: false
  ipv6: true
  vip: true
  subnets:
    subnet01:
      ipv6_subnet: 2001:db8:a::/64
      gateway_ipv6: 2001:db8:a::1
      ipv6_allocation_pools:
        - start: 2001:db8:a::0010
          end: 2001:db8:a::fff9
      enable_dhcp: false
      routes_ipv6:
        - destination: 2001:db8:b::/64
          nexthop: 2001:db8:a::1
      ipv6_address_mode: null
      ipv6_ra_mode: null
      vlan: 21
      physical_network: storage_subnet01  #optional, default: {{name.lower}}_{{subnet name}}
      network_type: flat                #optional, default: flat
      segmentation_id: 21                 #optional, default: undef
    subnet02:
      ipv6_subnet: 2001:db8:b::/64
      gateway_ipv6: 2001:db8:b::1
      ipv6_allocation_pools:
        - start: 2001:db8:b::0010
          end: 2001:db8:b::fff9
      enable_dhcp: false
      routes_ipv6:
        - destination: 2001:db8:a::/64
          nexthop: 2001:db8:b::1
      ipv6_address_mode: null
      ipv6_ra_mode: null
      vlan: 20
      physical_network: storage_subnet02  #optional, default: {{name.lower}}_{{subnet name}}
      network_type: flat                #optional, default: flat
      segmentation_id: 20                 #optional, default: undef
```

10.2. NETWORK ISOLATION

Red Hat OpenStack Platform (RHOSP) provides isolated overcloud networks so that you can host specific types of network traffic in isolation. Traffic is assigned to specific network interfaces or bonds. Using bonds provides fault tolerance and, if the correct bonding protocols are used, can also provide load sharing. If no isolated networks are configured, RHOSP uses the provisioning network for all services.

Network configuration consists of two parts: the parameters applied to the network as a whole, and the templates used to configure the network interfaces on the deployed nodes.

You can create the following isolated networks for your RHOSP deployment:

IPMI

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

Provisioning

Director uses this network for deployment and management. The provisioning network is normally configured on a dedicated interface. The initial deployment uses DHCP with PXE, then the network is converted to static IP. By default, PXE boot must occur on the native VLAN, although some system controllers allow booting from a VLAN. By default, the compute and storage nodes use the provisioning interface as their default gateway for DNS, NTP, and system maintenance.

Internal API

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

Tenant

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

- VLAN segregation, where each tenant network is a network VLAN.
- Tunneling (through VXLAN or GRE.

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

Storage

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

Storage Management

OpenStack Object Storage (swift) uses this network to synchronize data objects between participating replica nodes. The proxy service acts as the intermediary interface between user requests and the underlying storage layer. The proxy receives incoming requests and locates the necessary replica to retrieve the requested data. Services that use a Ceph backend connect over the Storage Management network, since they do not interact with Ceph directly but rather use the frontend service. Note that the RBD driver is an exception, as this traffic connects directly to Ceph.

External

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

Floating IP

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

NOTE

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

The undercloud can be used as a default gateway. However, all traffic is behind an IP masquerade NAT (Network Address Translation), and is not reachable from the rest of the RHOSP network. The undercloud is also a single point of failure for the overcloud default route. If there is an external gateway configured on a router device on the provisioning network, the undercloud neutron DHCP server can offer that service instead.

10.2.1. Networks required for each role

You can create tenant networks by using VLANs, but you can create VXLAN tunnels for special use without consuming tenant VLANs. It is possible to add VXLAN capability to a deployment with a tenant VLAN, but is it not possible to add a tenant VLAN to a deployed overcloud without serious disruption.

The following table details the isolated networks that are attached to each role:

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

10.2.2. Network definition file configuration options

Use the following tables to understand the available options for configuring your network definition file, **network_data.yaml**, in YAML format:

Table 10.1. Network data YAML options

Name	Option	Type	Default value
name	Name of the network.	string	
name_lower	Optional: Lower case name of the network.	string	**name.lower()**
dns_domain	Optional: DNS domain name for the network.	string	
mtu	Maximum Transmission Unit (MTU).	number	**1500**
ipv6	Optional: Set to true if using IPv6.	Boolean	**false**

Name	Option	Type	Default value
vip	Create a VIP on the network.	Boolean	**false**
subnets	Contains the subnets for the network.	dictionary	

Table 10.2. Network data YAML option for subnet definition

Name	Option	Type / Element	Example
ip_subnet	IPv4 CIDR block notation.	string	192.0.5.0/24
ipv6_subnet	IPv6 CIDR block notation.	string	2001:db6:fd00:1000::/64
gateway_ip	Optional: Gateway IPv4 address.	string	192.0.5.1
allocation_pools	Start and end address for the subnet.	list / dictionary	start: 192.0.5.100 end: 192.0.5.150
ipv6_allocation_pools	Start and end address for the subnet.	list / dictionary	start: 2001:db6:fd00:1000:100::1 end: 2001:db6:fd00:1000:150::1
routes	List of IPv4 networks that require routing through the network gateway.	list / dictionary	
routes_ipv6	List of IPv6 networks that require routing through the network gateway.	list / dictionary	
vlan	Optional: VLAN ID for the network.	number	

NOTE

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

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

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

Table 10.3. YAML data options for network virtual IPs

Name	Option	Type / Element	Default value
network	Neutron network name.	string	
ip_address	Optional: Pre-defined fixed IP address.	string	
subnet	Neutron subnet name. Specifies the subnet for the virtual IP neutron port. Required for deployments using routed networks.	string	
dns_name	Optional: FQDN (Fully Qualified Domain Name).	list / dictionary	**overcloud**
name	Optional: Virtual IP name.	string	**$network_name_virtual_ip**

10.2.3. Configuring network isolation

To enable and configure network isolation, you must add the required elements to the **network_data.yaml** configuration file.

Procedure

1. Create your network YAML definition file:

```
$ cp /usr/share/openstack-tripleo-heat-templates/network-data-samples/default-network-isolation-ipv6.yaml /home/stack/templates/network_data.yaml
```

2. Update the options in your **network_data.yaml** file to match the requirements for your overcloud networking environment:

```
- name: Storage
  name_lower: storage
  vip: true
  ipv6: true
  mtu: 1500
  subnets:
    storage_subnet:
      ipv6_subnet: fd00:fd00:fd00:3000::/64
      ipv6_allocation_pools:
        - start: fd00:fd00:fd00:3000::10
          end: fd00:fd00:fd00:3000:ffff:ffff:ffff:fffe
      vlan: 30
- name: StorageMgmt
  name_lower: storage_mgmt
  vip: true
  ipv6: true
  mtu: 1500
  subnets:
    storage_mgmt_subnet:
      ipv6_subnet: fd00:fd00:fd00:4000::/64
      ipv6_allocation_pools:
        - start: fd00:fd00:fd00:4000::10
          end: fd00:fd00:fd00:4000:ffff:ffff:ffff:fffe
      vlan: 40
- name: InternalApi
  name_lower: internal_api
  vip: true
  ipv6: true
  mtu: 1500
  subnets:
    internal_api_subnet:
      ipv6_subnet: fd00:fd00:fd00:2000::/64
      ipv6_allocation_pools:
        - start: fd00:fd00:fd00:2000::10
          end: fd00:fd00:fd00:2000:ffff:ffff:ffff:fffe
      vlan: 20
- name: Tenant
  name_lower: tenant
  vip: false # Tenant networks do not use VIPs
  ipv6: true
  mtu: 1500
  subnets:
    tenant_subnet:
      ipv6_subnet: fd00:fd00:fd00:5000::/64
      ipv6_allocation_pools:
        - start: fd00:fd00:fd00:5000::10
          end: fd00:fd00:fd00:5000:ffff:ffff:ffff:fffe
      vlan: 50
- name: External
  name_lower: external
  vip: true
  ipv6: true
  mtu: 1500
```

```
subnets:
  external_subnet:
    ipv6_subnet: 2001:db8:fd00:1000::/64
    ipv6_allocation_pools:
      - start: 2001:db8:fd00:1000::10
        end: 2001:db8:fd00:1000:ffff:ffff:ffff:fffe
    gateway_ipv6: 2001:db8:fd00:1000::1
    vlan: 10
```

10.3. COMPOSABLE NETWORKS

You can create custom composable networks if you want to host specific network traffic on different networks. Director provides a default network topology with network isolation enabled. You can find this configuration in the **/usr/share/openstack-tripleo-heat-templates/network-data-samples/default-network-isolation.yaml**.

The overcloud uses the following pre-defined set of network segments by default:

- Internal API
- Storage
- Storage management
- Tenant
- External

You can use composable networks to add networks for various services. For example, if you have a network that is dedicated to NFS traffic, you can present it to multiple roles.

Director supports the creation of custom networks during the deployment and update phases. You can use these additional networks for bare metal nodes, system management, or to create separate networks for different roles. You can also use them to create multiple sets of networks for split deployments where traffic is routed between networks.

10.3.1. Adding a composable network

Use composable networks to add networks for various services. For example, if you have a network that is dedicated to storage backup traffic, you can present the network to multiple roles.

You can find a sample file in the **/usr/share/openstack-tripleo-heat-templates/network-data-samples** directory.

Procedure

1. List the available sample configuration files:

```
$ ll /usr/share/openstack-tripleo-heat-templates/network-data-samples/
-rw-r--r--. 1 root root 1554 May 11 23:04 default-network-isolation-ipv6.yaml
-rw-r--r--. 1 root root 1181 May 11 23:04 default-network-isolation.yaml
-rw-r--r--. 1 root root 1126 May 11 23:04 ganesha-ipv6.yaml
-rw-r--r--. 1 root root 1100 May 11 23:04 ganesha.yaml
-rw-r--r--. 1 root root 3556 May 11 23:04 legacy-routed-networks-ipv6.yaml
-rw-r--r--. 1 root root 2929 May 11 23:04 legacy-routed-networks.yaml
```

```
-rw-r--r--. 1 root root  383 May 11 23:04 management-ipv6.yaml
-rw-r--r--. 1 root root  290 May 11 23:04 management.yaml
-rw-r--r--. 1 root root  136 May 11 23:04 no-networks.yaml
-rw-r--r--. 1 root root 2725 May 11 23:04 routed-networks-ipv6.yaml
-rw-r--r--. 1 root root 2033 May 11 23:04 routed-networks.yaml
-rw-r--r--. 1 root root  943 May 11 23:04 vip-data-default-network-isolation.yaml
-rw-r--r--. 1 root root  848 May 11 23:04 vip-data-fixed-ip.yaml
-rw-r--r--. 1 root root 1050 May 11 23:04 vip-data-routed-networks.yaml
```

2. Copy an example of a network configuration file which best suits your needs:

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

3. Edit your **network_data.yaml** configuration file and add a section for your new network:

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

You can use the following parameters in your **network_data.yaml** file:

name

Sets the name of the network.

vip

Enables the creation of a virtual IP address on the network.

name_lower

Sets the lowercase version of the name, which director maps to respective networks assigned to roles in the **roles_data.yaml** file.

subnets

One or more subnet defintions.

subnet_name

Sets the name of the subnet.

ip_subnet

Sets the IPv4 subnet in CIDR format.

allocation_pools

Sets the IP range for the IPv4 subnet.

gateway_ip

Sets the gateway for the network.

vlan

Sets the VLAN ID for the network.

ipv6

Set the value to true or false.

ipv6_subnet

Sets the IPv6 subnet.

gateway_ipv6

Sets the gateway for the IPv6 network.

ipv6_allocation_pools

Sets the IP range for the IPv6 subnet.

routes_ipv6

Sets the routes for the IPv6 network.

4. Copy the sample network VIP definition template you require from **/usr/share/openstack-tripleo-heat-templates/network-data-samples** to your environment file directory. The following example copies the **vip-data-default-network-isolation.yaml** to a local environment file named **vip_data.yaml**:

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

5. Edit your **vip_data.yaml** configuration file. The virtual IP data is a list of virtual IP address definitions, each containing the name of the network where the IP address is allocated.

```
- network: storage_mgmt
  dns_name: overcloud
- network: internal_api
  dns_name: overcloud
- network: storage
  dns_name: overcloud
- network: external
  dns_name: overcloud
  ip_address: <vip_address>
- network: ctlplane
  dns_name: overcloud
```

- Replace **<vip_address>** with the required virtual IP address.

You can use the following parameters in your **vip_data.yaml** file:

network

Sets the neutron network name. This is the only required parameter.

ip_address

Sets the IP address of the VIP.

subnet

Sets the neutron subnet name. Use to specify the subnet when creating the virtual IP neutron port. This parameter is required when your deployment uses routed networks.

dns_name

Sets the FQDN (Fully Qualified Domain Name).

name

Sets the virtual IP name.

6. Copy a sample network configuration template. Jinja2 templates are used to define NIC

configuration templates. Browse the examples provided in the **/usr/share/ansible/roles/tripleo_network_config/templates/** directory, if one of the examples matches your requirements, use it. If the examples do not match your requirements, copy a sample configuration file, and modify it for your needs:

```
$ cp
/usr/share/ansible/roles/tripleo_network_config/templates/single_nic_vlans/single_nic_vlans.j2
/home/stack/templates/
```

7. Edit your **single_nic_vlans.j2** configuration file:

```
---
{% set mtu_list = [ctlplane_mtu] %}
{% for network in role_networks %}
{{ mtu_list.append(lookup('vars', networks_lower[network] ~ '_mtu')) }}
{%- endfor %}
{% set min_viable_mtu = mtu_list | max %}
network_config:
- type: ovs_bridge
  name: {{ neutron_physical_bridge_name }}
  mtu: {{ min_viable_mtu }}
  use_dhcp: false
  dns_servers: {{ ctlplane_dns_nameservers }}
  domain: {{ dns_search_domains }}
  addresses:
  - ip_netmask: {{ ctlplane_ip }}/{{ ctlplane_subnet_cidr }}
  routes: {{ ctlplane_host_routes }}
  members:
  - type: interface
    name: nic1
    mtu: {{ min_viable_mtu }}
    # force the MAC address of the bridge to this interface
    primary: true
{% for network in role_networks %}
  - type: vlan
    mtu: {{ lookup('vars', networks_lower[network] ~ '_mtu') }}
    vlan_id: {{ lookup('vars', networks_lower[network] ~ '_vlan_id') }}
    addresses:
    - ip_netmask:
        {{ lookup('vars', networks_lower[network] ~ '_ip') }}/{{ lookup('vars',
networks_lower[network] ~ '_cidr') }}
    routes: {{ lookup('vars', networks_lower[network] ~ '_host_routes') }}
{% endfor %}
```

8. Set the **network_config** template in **overcloud-baremetal-deploy.yaml** configuration file:

```
- name: CephStorage
  count: 3
  defaults:
    networks:
    - network: storage
    - network: storage_mgmt
    - network: storage_backup
    network_config:
      template: /home/stack/templates/single_nic_vlans.j2
```

9. Provision the overcloud networks. This action generates an output file which will be used an an environment file when deploying the overcloud:

```
(undercloud)$ openstack overcloud network provision --output <deployment_file>
/home/stack/templates/<networks_definition_file>.yaml
```

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

10. Provision the network VIPs and generate the **vip-deployed-environment.yaml** file. You use this file when you deploy the overcloud:

```
(overcloud)$ openstack overcloud network vip provision  --stack <stack> --output
<deployment_file> /home/stack/templates/vip_data.yaml
```

   - Replace **<stack>** with the name of the stack for which the network VIPs are provisioned. If not specified, the default is overcloud.
   - Replace **<deployment_file>** with the name of the heat environment file to generate for inclusion in the deployment command, for example **/home/stack/templates/overcloud-vip-deployed.yaml**.

10.3.2. Including a composable network in a role

You can assign composable networks to the overcloud roles defined in your environment. For example, you might include a custom **StorageBackup** network with your Ceph Storage nodes.

Procedure

1. If you do not already have a custom **roles_data.yaml** file, copy the default to your home directory:

```
$ cp /usr/share/openstack-tripleo-heat-templates/roles_data.yaml
/home/stack/templates/roles_data.yaml
```

2. Edit the custom **roles_data.yaml** file.
3. Include the network name in the **networks** list for the role that you want to add the network to. For example, to add the **StorageBackup** network to the Ceph Storage role, use the following example snippet:

```
- name: CephStorage
  description: |
    Ceph OSD Storage node role
  networks:
    Storage
      subnet: storage_subnet
    StorageMgmt
      subnet: storage_mgmt_subnet
    StorageBackup
      subnet: storage_backup_subnet
```

-

4. After you add custom networks to their respective roles, save the file.

When you run the **openstack overcloud deploy** command, include the custom **roles_data.yaml** file using the **-r** option. Without the **-r** option, the deployment command uses the default set of roles with their respective assigned networks.

10.3.3. Assigning OpenStack services to composable networks

Each OpenStack service is assigned to a default network type in the resource registry. These services are bound to IP addresses within the network type's assigned network. Although the OpenStack services are divided among these networks, the number of actual physical networks can differ as defined in the network environment file. You can reassign OpenStack services to different network types by defining a new network map in an environment file, for example, **/home/stack/templates/service-reassignments.yaml**. The **ServiceNetMap** parameter determines the network types that you want to use for each service.

For example, you can reassign the Storage Management network services to the Storage Backup Network by modifying the highlighted sections:

```
parameter_defaults:
  ServiceNetMap:
    SwiftStorageNetwork: storage_backup
    CephClusterNetwork: storage_backup
```

Changing these parameters to **storage_backup** places these services on the Storage Backup network instead of the Storage Management network. This means that you must define a set of **parameter_defaults** only for the Storage Backup network and not the Storage Management network.

Director merges your custom **ServiceNetMap** parameter definitions into a pre-defined list of defaults that it obtains from **ServiceNetMapDefaults** and overrides the defaults. Director returns the full list, including customizations, to **ServiceNetMap**, which is used to configure network assignments for various services.

Service mappings apply to networks that use **vip: true** in the **network_data.yaml** file for nodes that use Pacemaker. The overcloud load balancer redirects traffic from the VIPs to the specific service endpoints.

NOTE

You can find a full list of default services in the **ServiceNetMapDefaults** parameter in the **/usr/share/openstack-tripleo-heat-templates/network/service_net_map.j2.yaml** file.

10.3.4. Enabling custom composable networks

Use one of the default NIC templates to enable custom composable networks. In this example, use the Single NIC with VLANs template, (**custom_single_nic_vlans**).

Procedure

1. Source the stackrc undercloud credential file:

```
$ source ~/stackrc
```

2. Provision the overcloud networks:

```
$ openstack overcloud network provision \
  --output overcloud-networks-deployed.yaml \
  custom_network_data.yaml
```

3. Provision the network VIPs:

```
$ openstack overcloud network vip provision \
    --stack overcloud \
    --output overcloud-networks-vips-deployed.yaml \
     custom_vip_data.yaml
```

4. Provision the overcloud nodes:

```
$ openstack overcloud node provision \
    --stack overcloud \
    --output overcloud-baremetal-deployed.yaml \
    overcloud-baremetal-deploy.yaml
```

5. Construct your **openstack overcloud deploy** command, specifying the configuration files and templates in the required order, for example:

```
$ openstack overcloud deploy --templates \
  --networks-file network_data_v2.yaml \
  -e overcloud-networks-deployed.yaml \
  -e overcloud-networks-vips-deployed.yaml \
  -e overcloud-baremetal-deployed.yaml
  -e custom-net-single-nic-with-vlans.yaml
```

This example command deploys the composable networks, including your additional custom networks, across nodes in your overcloud.

10.3.5. Renaming the default networks

You can use the **network_data.yaml** file to modify the user-visible names of the default networks:

- InternalApi
- External
- Storage
- StorageMgmt
- Tenant

To change these names, do not modify the **name** field. Instead, change the **name_lower** field to the new name for the network and update the ServiceNetMap with the new name.

Procedure

1. In your **network_data.yaml** file, enter new names in the **name_lower** parameter for each network that you want to rename:

```
- name: InternalApi
  name_lower: MyCustomInternalApi
```

2. Include the default value of the **name_lower** parameter in the **service_net_map_replace** parameter:

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

10.4. CUSTOM NETWORK INTERFACE TEMPLATES

After you configure Section 10.2, "Network isolation", you can create a set of custom network interface templates to suit the nodes in your environment. For example, you can include the following files:

- The environment file to configure network defaults (**/usr/share/openstack-tripleo-heat-templates/environments/network/multiple-nics/network-environment.yaml**).
- Templates to define your NIC layout for each node. The overcloud core template collection contains a set of defaults for different use cases. To create a custom NIC template, render a default Jinja2 template as the basis for your custom templates.
- A custom environment file to enable NICs. This example uses a custom environment file (**/home/stack/templates/custom-network-configuration.yaml**) that references your custom interface templates.
- Any additional environment files to customize your networking parameters.
- If you customize your networks, a custom **network_data.yaml** file.
- If you create additional or custom composable networks, a custom **network_data.yaml** file and a custom **roles_data.yaml** file.

NOTE

Some of the files in the previous list are Jinja2 format files and have a **.j2.yaml** extension. Director renders these files to **.yaml** versions during deployment.

10.4.1. Custom network architecture

The example NIC templates might not suit a specific network configuration. For example, you might want to create your own custom NIC template that suits a specific network layout. You might want to separate the control services and data services on to separate NICs. In this situation, you can map the service to NIC assignments in the following way:

- NIC1 (Provisioning)
  - Provisioning / Control Plane
- NIC2 (Control Group)
  - Internal API
  - Storage Management

- External (Public API)
- NIC3 (Data Group)
    - Tenant Network (VXLAN tunneling)
    - Tenant VLANs / Provider VLANs
    - Storage
    - External VLANs (Floating IP/SNAT)
- NIC4 (Management)
    - Management

10.4.2. Network interface reference

The network interface configuration contains the following parameters:

Interface

Defines a single network interface. The configuration defines each interface using either the actual interface name ("eth0", "eth1", "enp0s25") or a set of numbered interfaces ("nic1", "nic2", "nic3"):

```
- type: interface
  name: nic2
```

Table 10.4. interface options

Option	Default	Description
name		Name of the interface.
use_dhcp	False	Use DHCP to get an IP address.
use_dhcpv6	False	Use DHCP to get a v6 IP address.
addresses		A list of IP addresses assigned to the interface.
routes		A list of routes assigned to the interface. For more information, see routes.
mtu	1500	The maximum transmission unit (MTU) of the connection.
primary	False	Defines the interface as the primary interface.

Option	Default	Description
persist_mapping	False	Write the device alias configuration instead of the system names.
dhclient_args	None	Arguments that you want to pass to the DHCP client.
dns_servers	None	List of DNS servers that you want to use for the interface.
ethtool_opts		Set this option to **"rx-flow-hash udp4 sdfn"** to improve throughput when you use VXLAN on certain NICs.

vlan

Defines a VLAN. Use the VLAN ID and subnet passed from the **parameters** section.

For example:

```
- type: vlan
  device: nic{{ loop.index + 1 }}
  mtu: {{ lookup('vars', networks_lower[network] ~ '_mtu') }}
  vlan_id: {{ lookup('vars', networks_lower[network] ~ '_vlan_id') }}
  addresses:
  - ip_netmask:
    {{ lookup('vars', networks_lower[network] ~ '_ip') }}/{{ lookup('vars', networks_lower[network] ~
'_cidr') }}
  routes: {{ lookup('vars', networks_lower[network] ~ '_host_routes') }}
```

Table 10.5. vlan options

Option	Default	Description
vlan_id		The VLAN ID.
device		The parent device to attach the VLAN. Use this parameter when the VLAN is not a member of an OVS bridge. For example, use this parameter to attach the VLAN to a bonded interface device.
use_dhcp	False	Use DHCP to get an IP address.
use_dhcpv6	False	Use DHCP to get a v6 IP address.

Option	Default	Description
addresses		A list of IP addresses assigned to the VLAN.
routes		A list of routes assigned to the VLAN. For more information, see routes.
mtu	1500	The maximum transmission unit (MTU) of the connection.
primary	False	Defines the VLAN as the primary interface.
persist_mapping	False	Write the device alias configuration instead of the system names.
dhclient_args	None	Arguments that you want to pass to the DHCP client.
dns_servers	None	List of DNS servers that you want to use for the VLAN.

ovs_bond

Defines a bond in Open vSwitch to join two or more **interfaces** together. This helps with redundancy and increases bandwidth.

For example:

```
members:
  - type: ovs_bond
    name: bond1
    mtu: {{ min_viable_mtu }}
    ovs_options: {{ bond_interface_ovs_options }}
    members:
    - type: interface
      name: nic2
      mtu: {{ min_viable_mtu }}
      primary: true
    - type: interface
      name: nic3
      mtu: {{ min_viable_mtu }}
```

Table 10.6. ovs_bond options

Option	Default	Description
name		Name of the bond.
use_dhcp	False	Use DHCP to get an IP address.
use_dhcpv6	False	Use DHCP to get a v6 IP address.
addresses		A list of IP addresses assigned to the bond.
routes		A list of routes assigned to the bond. For more information, see routes.
mtu	1500	The maximum transmission unit (MTU) of the connection.
primary	False	Defines the interface as the primary interface.
members		A sequence of interface objects that you want to use in the bond.
ovs_options		A set of options to pass to OVS when creating the bond.
ovs_extra		A set of options to set as the OVS_EXTRA parameter in the network configuration file of the bond.
defroute	True	Use a default route provided by the DHCP service. Only applies when you enable **use_dhcp** or **use_dhcpv6**.
persist_mapping	False	Write the device alias configuration instead of the system names.
dhclient_args	None	Arguments that you want to pass to the DHCP client.
dns_servers	None	List of DNS servers that you want to use for the bond.

ovs_bridge

Defines a bridge in Open vSwitch, which connects multiple **interface**, **ovs_bond**, and **vlan** objects together.

The network interface type, **ovs_bridge**, takes a parameter **name**.

NOTE

If you have multiple bridges, you must use distinct bridge names other than accepting the default name of **bridge_name**. If you do not use distinct names, then during the converge phase, two network bonds are placed on the same bridge.

If you are defining an OVS bridge for the external tripleo network, then retain the values **bridge_name** and **interface_name** as your deployment framework automatically replaces these values with an external bridge name and an external interface name, respectively.

For example:

```
- type: ovs_bridge
  name: br-bond
  dns_servers: {{ ctlplane_dns_nameservers }}
  domain: {{ dns_search_domains }}
  members:
  - type: ovs_bond
    name: bond1
    mtu: {{ min_viable_mtu }}
    ovs_options: {{ bound_interface_ovs_options }}
    members:
    - type: interface
      name: nic2
      mtu: {{ min_viable_mtu }}
      primary: true
    - type: interface
      name: nic3
      mtu: {{ min_viable_mtu }}
```

NOTE

The OVS bridge connects to the Networking service (neutron) server to obtain configuration data. If the OpenStack control traffic, typically the Control Plane and Internal API networks, is placed on an OVS bridge, then connectivity to the neutron server is lost whenever you upgrade OVS, or the OVS bridge is restarted by the admin user or process. This causes some downtime. If downtime is not acceptable in these circumstances, then you must place the Control group networks on a separate interface or bond rather than on an OVS bridge:

- You can achieve a minimal setting when you put the Internal API network on a VLAN on the provisioning interface and the OVS bridge on a second interface.
- To implement bonding, you need at least two bonds (four network interfaces). Place the control group on a Linux bond (Linux bridge). If the switch does not support LACP fallback to a single interface for PXE boot, then this solution requires at least five NICs.

Table 10.7. ovs_bridge options

Option	Default	Description
name		Name of the bridge.
use_dhcp	False	Use DHCP to get an IP address.
use_dhcpv6	False	Use DHCP to get a v6 IP address.
addresses		A list of IP addresses assigned to the bridge.
routes		A list of routes assigned to the bridge. For more information, see routes.
mtu	1500	The maximum transmission unit (MTU) of the connection.
members		A sequence of interface, VLAN, and bond objects that you want to use in the bridge.
ovs_options		A set of options to pass to OVS when creating the bridge.
ovs_extra		A set of options to to set as the OVS_EXTRA parameter in the network configuration file of the bridge.
defroute	True	Use a default route provided by the DHCP service. Only applies when you enable **use_dhcp** or **use_dhcpv6**.
persist_mapping	False	Write the device alias configuration instead of the system names.
dhclient_args	None	Arguments that you want to pass to the DHCP client.
dns_servers	None	List of DNS servers that you want to use for the bridge.

linux_bond

Defines a Linux bond that joins two or more **interfaces** together. This helps with redundancy and increases bandwidth. Ensure that you include the kernel-based bonding options in the **bonding_options** parameter.

For example:

```
- type: linux_bridge
  name: {{ neutron_physical_bridge_name }}
  mtu: {{ min_viable_mtu }}
  use_dhcp: false
  dns_servers: {{ ctlplane_dns_nameservers }}
  domain: {{ dns_search_domains }}
  addresses:
  - ip_netmask: {{ ctlplane_ip }}/{{ ctlplane_subnet_cidr }}
  routes: {{ ctlplane_host_routes }}
```

Note that **nic2** uses **primary: true** to ensure that the bond uses the MAC address for **nic2**.

Table 10.8. linux_bond options

Option	Default	Description
name		Name of the bond.
use_dhcp	False	Use DHCP to get an IP address.
use_dhcpv6	False	Use DHCP to get a v6 IP address.
addresses		A list of IP addresses assigned to the bond.
routes		A list of routes assigned to the bond. See routes.
mtu	1500	The maximum transmission unit (MTU) of the connection.
primary	False	Defines the interface as the primary interface.
members		A sequence of interface objects that you want to use in the bond.
bonding_options		A set of options when creating the bond.
defroute	True	Use a default route provided by the DHCP service. Only applies when you enable **use_dhcp** or **use_dhcpv6**.
persist_mapping	False	Write the device alias configuration instead of the system names.

Option	Default	Description
dhclient_args	None	Arguments that you want to pass to the DHCP client.
dns_servers	None	List of DNS servers that you want to use for the bond.

linux_bridge

Defines a Linux bridge, which connects multiple **interface**, **linux_bond**, and **vlan** objects together. The external bridge also uses two special values for parameters:

- **bridge_name**, which is replaced with the external bridge name.
- **interface_name**, which is replaced with the external interface.

For example:

```
- type: linux_bridge
  name: bridge_name
  mtu:
   get_attr: [MinViableMtu, value]
  use_dhcp: false
  dns_servers:
   get_param: DnsServers
  domain:
   get_param: DnsSearchDomains
  addresses:
  - ip_netmask:
    list_join:
    - /
    - - get_param: ControlPlaneIp
     - get_param: ControlPlaneSubnetCidr
  routes:
   list_concat_unique:
    - get_param: ControlPlaneStaticRoutes
```

Table 10.9. linux_bridge options

Option	Default	Description
name		Name of the bridge.
use_dhcp	False	Use DHCP to get an IP address.
use_dhcpv6	False	Use DHCP to get a v6 IP address.
addresses		A list of IP addresses assigned to the bridge.

Option	Default	Description
routes		A list of routes assigned to the bridge. For more information, see routes.
mtu	1500	The maximum transmission unit (MTU) of the connection.
members		A sequence of interface, VLAN, and bond objects that you want to use in the bridge.
defroute	True	Use a default route provided by the DHCP service. Only applies when you enable **use_dhcp** or **use_dhcpv6**.
persist_mapping	False	Write the device alias configuration instead of the system names.
dhclient_args	None	Arguments that you want to pass to the DHCP client.
dns_servers	None	List of DNS servers that you want to use for the bridge.

routes

Defines a list of routes to apply to a network interface, VLAN, bridge, or bond.

For example:

```
- type: linux_bridge
  name: bridge_name
  ...
  routes: {{ [ctlplane_host_routes] | flatten | unique }}
```

Option	Default	Description
ip_netmask	None	IP and netmask of the destination network.
default	False	Sets this route to a default route. Equivalent to setting **ip_netmask: 0.0.0.0/0**.

Option	Default	Description
next_hop	None	The IP address of the router used to reach the destination network.

10.4.3. Example network interface layout

The following snippet for an example controller node NIC template demonstrates how to configure the custom network scenario to keep the control group separate from the OVS bridge:

```
network_config:
- type: interface
  name: nic1
  mtu: {{ ctlplane_mtu }}
  use_dhcp: false
  addresses:
  - ip_netmask: {{ ctlplane_ip }}/{{ ctlplane_subnet_cidr }}
  routes: {{ ctlplane_host_routes }}
- type: linux_bond
  name: bond_api
  mtu: {{ min_viable_mtu_ctlplane }}
  use_dhcp: false
  bonding_options: {{ bond_interface_ovs_options }}
  dns_servers: {{ ctlplane_dns_nameservers }}
  domain: {{ dns_search_domains }}
  members:
  - type: interface
    name: nic2
    mtu: {{ min_viable_mtu_ctlplane }}
    primary: true
  - type: interface
    name: nic3
    mtu: {{ min_viable_mtu_ctlplane }}
{% for network in role_networks if not network.startswith('Tenant') %}
- type: vlan
  device: bond_api
  mtu: {{ lookup('vars', networks_lower[network] ~ '_mtu') }}
  vlan_id: {{ lookup('vars', networks_lower[network] ~ '_vlan_id') }}
  addresses:
  - ip_netmask: {{ lookup('vars', networks_lower[network] ~ '_ip') }}/{{ lookup('vars',
networks_lower[network] ~ '_cidr') }}
  routes: {{ lookup('vars', networks_lower[network] ~ '_host_routes') }}
{% endfor %}
- type: ovs_bridge
  name: {{ neutron_physical_bridge_name }}
  dns_servers: {{ ctlplane_dns_nameservers }}
  members:
  - type: linux_bond
    name: bond-data
    mtu: {{ min_viable_mtu_dataplane }}
    bonding_options: {{ bond_interface_ovs_options }}
    members:
    - type: interface
      name: nic4
      mtu: {{ min_viable_mtu_dataplane }}
      primary: true
    - type: interface
      name: nic5
      mtu: {{ min_viable_mtu_dataplane }}
{% for network in role_networks if network.startswith('Tenant') %}
  - type: vlan
    device: bond-data
    mtu: {{ lookup('vars', networks_lower[network] ~ '_mtu') }}
    vlan_id: {{ lookup('vars', networks_lower[network] ~ '_vlan_id') }}
    addresses:
    - ip_netmask: {{ lookup('vars', networks_lower[network] ~ '_ip') }}/{{ lookup('vars',
networks_lower[network] ~ '_cidr') }}
    routes: {{ lookup('vars', networks_lower[network] ~ '_host_routes') }}
```

This template uses five network interfaces and assigns a number of tagged VLAN devices to the numbered interfaces. On **nic4** and **nic5**, this template creates the OVS bridges.

10.5. ADDITIONAL OVERCLOUD NETWORK CONFIGURATION

This chapter follows on from the concepts and procedures outlined in Section 10.4, “Custom network interface templates” and provides some additional information to help configure parts of your overcloud network.

10.5.1. Configuring custom interfaces

Individual interfaces might require modification. The following example shows the modifications that are necessary to use a second NIC to connect to an infrastructure network with DHCP addresses, and to use another NIC for the bond:

```
network_config:
  # Add a DHCP infrastructure network to nic2
  - type: interface
    name: nic2
    mtu: {{ tenant_mtu }}
    use_dhcp: true
    primary: true
  - type: vlan
    mtu: {{ tenant_mtu }}
    vlan_id: {{ tenant_vlan_id }}
    addresses:
    - ip_netmask: {{ tenant_ip }}/{{ tenant_cidr }}
    routes: {{ [tenant_host_routes] | flatten | unique }}
  - type: ovs_bridge
    name: br-bond
    mtu: {{ external_mtu }}
    dns_servers: {{ ctlplane_dns_nameservers }}
    use_dhcp: false
    members:
      - type: interface
        name: nic10
        mtu: {{ external_mtu }}
        use_dhcp: false
        primary: true
      - type: vlan
```

```
        mtu: {{ external_mtu }}
        vlan_id: {{ external_vlan_id }}
        addresses:
        - ip_netmask: {{ external_ip }}/{{ external_cidr }}
        routes: {{ [external_host_routes, [{'default': True, 'next_hop': external_gateway_ip}]] | flatten |
unique }}
```

The network interface template uses either the actual interface name (**eth0**, **eth1**, **enp0s25**) or a set of numbered interfaces (**nic1**, **nic2**, **nic3**). The network interfaces of hosts within a role do not have to be exactly the same when you use numbered interfaces (**nic1**, **nic2**, etc.) instead of named interfaces (**eth0**, **eno2**, etc.). For example, one host might have interfaces **em1** and **em2**, while another has **eno1** and **eno2**, but you can refer to the NICs of both hosts as **nic1** and **nic2**.

The order of numbered interfaces corresponds to the order of named network interface types:

- **ethX** interfaces, such as **eth0**, **eth1**, etc. These are usually onboard interfaces.
- **enoX** interfaces, such as **eno0**, **eno1**, etc. These are usually onboard interfaces.
- **enX** interfaces, sorted alpha numerically, such as **enp3s0**, **enp3s1**, **ens3**, etc. These are usually add-on interfaces.

The numbered NIC scheme includes only live interfaces, for example, if the interfaces have a cable attached to the switch. If you have some hosts with four interfaces and some with six interfaces, use **nic1** to **nic4** and attach only four cables on each host.

Customizing NIC mappings for pre-provisioned nodes

If you are using pre-provisioned nodes, you can specify **os-net-config** mappings for specific nodes by configuring the **NetConfigDataLookup** heat parameter in an environment file.

NOTE

The configuration of the **NetConfigDataLookup** heat parameter is equivalent to the **net_config_data_lookup** property in your node definition file, **overcloud-baremetal-deploy.yaml**. If you are not using pre-provisioned nodes, you must configure the NIC mappings in your node definition file. For more information on configuring the **net_config_data_lookup** property, see Bare-metal node provisioning attributes.

You can assign aliases to the physical interfaces on each node to pre-determine which physical NIC maps to specific aliases, such as **nic1** or **nic2**, and you can map a MAC address to a specified alias. You can map specific nodes by using the MAC address or DMI keyword, or you can map a group of nodes by using a DMI keyword. The following example configures three nodes and two node groups with aliases to the physical interfaces. The resulting configuration is applied by **os-net-config**. On each node, you can see the applied configuration in the **interface_mapping** section of the **/etc/os-net-config/mapping.yaml** file.

Example os-net-config-mappings.yaml

```
NetConfigDataLookup:
  node1: (1)
    nic1: "00:c8:7c:e6:f0:2e"
  node2:
    nic1: "00:18:7d:99:0c:b6"
  node3: (2)
```

```
      dmiString: "system-uuid" (3)
      id: 'A8C85861-1B16-4803-8689-AFC62984F8F6'
      nic1: em3
    # Dell PowerEdge
    nodegroup1: (4)
      dmiString: "system-product-name"
      id: "PowerEdge R630"
      nic1: em3
      nic2: em1
      nic3: em2
    # Cisco UCS B200-M4"
    nodegroup2:
      dmiString: "system-product-name"
      id: "UCSB-B200-M4"
      nic1: enp7s0
      nic2: enp6s0
```

1. Maps **node1** to the specified MAC address, and assigns **nic1** as the alias for the MAC address on this node.

2. Maps **node3** to the node with the system UUID "A8C85861-1B16-4803-8689-AFC62984F8F6", and assigns **nic1** as the alias for **em3** interface on this node.

3. The **dmiString** parameter must be set to a valid string keyword. For a list of the valid string keywords, see the DMIDECODE(8) man page.

4. Maps all the nodes in **nodegroup1** to nodes with the product name "PowerEdge R630", and assigns **nic1**, **nic2**, and **nic3** as the alias for the named interfaces on these nodes.

NOTE

Normally, **os-net-config** registers only the interfaces that are already connected in an **UP** state. However, if you hardcode interfaces with a custom mapping file, the interface is registered even if it is in a **DOWN** state.

10.5.2. Configuring routes and default routes

You can set the default route of a host in one of two ways. If the interface uses DHCP and the DHCP server offers a gateway address, the system uses a default route for that gateway. Otherwise, you can set a default route on an interface with a static IP.

Although the Linux kernel supports multiple default gateways, it uses only the gateway with the lowest metric. If there are multiple DHCP interfaces, this can result in an unpredictable default gateway. In this case, it is recommended to set **defroute: false** for interfaces other than the interface that uses the default route.

For example, you might want a DHCP interface (**nic3**) to be the default route. Use the following YAML snippet to disable the default route on another DHCP interface (**nic2**):

```
# No default route on this DHCP interface
- type: interface
  name: nic2
  use_dhcp: true
  defroute: false
```

```
# Instead use this DHCP interface as the default route
- type: interface
  name: nic3
  use_dhcp: true
```

NOTE

The **defroute** parameter applies only to routes obtained through DHCP.

To set a static route on an interface with a static IP, specify a route to the subnet. For example, you can set a route to the 10.1.2.0/24 subnet through the gateway at 172.17.0.1 on the Internal API network:

```
    - type: vlan
      device: bond1
      vlan_id: 9
      addresses:
      - ip_netmask: 172.17.0.100/16
      routes:
      - ip_netmask: 10.1.2.0/24
        next_hop: 172.17.0.1
```

10.5.3. Configuring policy-based routing

To configure unlimited access from different networks on Controller nodes, configure policy-based routing. Policy-based routing uses route tables where, on a host with multiple interfaces, you can send traffic through a particular interface depending on the source address. You can route packets that come from different sources to different networks, even if the destinations are the same.

For example, you can configure a route to send traffic to the Internal API network, based on the source address of the packet, even when the default route is for the External network. You can also define specific route rules for each interface.

Red Hat OpenStack Platform uses the **os-net-config** tool to configure network properties for your overcloud nodes. The **os-net-config** tool manages the following network routing on Controller nodes:

- Routing tables in the **/etc/iproute2/rt_tables** file
- IPv4 rules in the **/etc/sysconfig/network-scripts/rule-{ifname}** file
- IPv6 rules in the **/etc/sysconfig/network-scripts/rule6-{ifname}** file
- Routing table specific routes in the **/etc/sysconfig/network-scripts/route-{ifname}**

Prerequisites

- You have installed the undercloud successfully. For more information, see Installing director in the *Director Installation and Usage* guide.

Procedure

1. Create the **interface** entries in a custom NIC template from the **/home/stack/templates/custom-nics** directory, define a route for the interface, and define rules that are relevant to your deployment:

```
network_config:
- type: interface
  name: em1
  use_dhcp: false
  addresses:
    - ip_netmask: {{ external_ip }}/{{ external_cidr}}
  routes:
    - default: true
      next_hop: {{ external_gateway_ip }}
    - ip_netmask: {{ external_ip }}/{{ external_cidr}}
      next_hop: {{ external_gateway_ip }}
      route_table: 2
      route_options: metric 100
  rules:
    - rule: "iif em1 table 200"
      comment: "Route incoming traffic to em1 with table 200"
    - rule: "from 192.0.2.0/24 table 200"
      comment: "Route all traffic from 192.0.2.0/24 with table 200"
    - rule: "add blackhole from 172.19.40.0/24 table 200"
    - rule: "add unreachable iif em1 from 192.168.1.0/24"
```

2. Include your custom NIC configuration and network environment files in the deployment command, along with any other environment files relevant to your deployment:

```
$ openstack overcloud deploy --templates \
-e /home/stack/templates/<custom-nic-template>
-e <OTHER_ENVIRONMENT_FILES>
```

Verification

- Enter the following commands on a Controller node to verify that the routing configuration is functioning correctly:

```
$ cat /etc/iproute2/rt_tables
$ ip route
$ ip rule
```

10.5.4. Configuring jumbo frames

The Maximum Transmission Unit (MTU) setting determines the maximum amount of data transmitted with a single Ethernet frame. Using a larger value results in less overhead because each frame adds data in the form of a header. The default value is 1500 and using a higher value requires the configuration of the switch port to support jumbo frames. Most switches support an MTU of at least 9000, but many are configured for 1500 by default.

The MTU of a VLAN cannot exceed the MTU of the physical interface. Ensure that you include the MTU value on the bond or interface.

The Storage, Storage Management, Internal API, and Tenant networks can all benefit from jumbo frames.

You can alter the value of the **mtu** in the **jinja2** template or in the **network_data.yaml** file. If you set the value in the **network_data.yaml** file it is rendered during deployment.

WARNING

Routers typically cannot forward jumbo frames across Layer 3 boundaries. To avoid connectivity issues, do not change the default MTU for the Provisioning interface, External interface, and any Floating IP interfaces.

```
---
{% set mtu_list = [ctlplane_mtu] %}
{% for network in role_networks %}
{{ mtu_list.append(lookup('vars', networks_lower[network] ~ '_mtu')) }}
{%- endfor %}
{% set min_viable_mtu = mtu_list | max %}
network_config:
- type: ovs_bridge
  name: bridge_name
  mtu: {{ min_viable_mtu }}
  use_dhcp: false
  dns_servers: {{ ctlplane_dns_nameservers }}
  domain: {{ dns_search_domains }}
  addresses:
  - ip_netmask: {{ ctlplane_ip }}/{{ ctlplane_subnet_cidr }}
  routes: {{ [ctlplane_host_routes] | flatten | unique }}
  members:
  - type: interface
    name: nic1
    mtu: {{ min_viable_mtu }}
    primary: true
  - type: vlan
    mtu: 9000  1
    vlan_id: {{ storage_vlan_id }}
    addresses:
    - ip_netmask: {{ storage_ip }}/{{ storage_cidr }}
    routes: {{ [storage_host_routes] | flatten | unique }}
  - type: vlan
    mtu: {{ storage_mgmt_mtu }}  2
    vlan_id: {{ storage_mgmt_vlan_id }}
    addresses:
    - ip_netmask: {{ storage_mgmt_ip }}/{{ storage_mgmt_cidr }}
    routes: {{ [storage_mgmt_host_routes] | flatten | unique }}
  - type: vlan
    mtu: {{ internal_api_mtu }}
    vlan_id: {{ internal_api_vlan_id }}
    addresses:
    - ip_netmask: {{ internal_api_ip }}/{{ internal_api_cidr }}
    routes: {{ [internal_api_host_routes] | flatten | unique }}
  - type: vlan
    mtu: {{ tenant_mtu }}
    vlan_id: {{ tenant_vlan_id }}
    addresses:
    - ip_netmask: {{ tenant_ip }}/{{ tenant_cidr }}
    routes: {{ [tenant_host_routes] | flatten | unique }}
  - type: vlan
    mtu: {{ external_mtu }}
    vlan_id: {{ external_vlan_id }}
    addresses:
    - ip_netmask: {{ external_ip }}/{{ external_cidr }}
    routes: {{ [external_host_routes, [{'default': True, 'next_hop': external_gateway_ip}]] | flatten |
unique }}
```

1. mtu value updated directly in the **jinja2** template.
2. mtu value is taken from the **network_data.yaml** file during deployment.

10.5.5. Configuring ML2/OVN northbound path MTU discovery for jumbo frame fragmentation

If a VM on your internal network sends jumbo frames to an external network, and the maximum transmission unit (MTU) of the internal network exceeds the MTU of the external network, a northbound frame can easily exceed the capacity of the external network.

ML2/OVS automatically handles this oversized packet issue, and ML2/OVN handles it automatically for TCP packets.

But to ensure proper handling of oversized northbound UDP packets in a deployment that uses the ML2/OVN mechanism driver, you need to perform additional configuration steps.

These steps configure ML2/OVN routers to return ICMP "fragmentation needed" packets to the sending VM, where the sending application can break the payload into smaller packets.

NOTE

In east/west traffic, a RHOSP ML2/OVN deployment does not support fragmentation of packets that are larger than the smallest MTU on the east/west path. For example:

- VM1 is on Network1 with an MTU of 1300.
- VM2 is on Network2 with an MTU of 1200.
- A ping in either direction between VM1 and VM2 with a size of 1171 or less succeeds. A ping with a size greater than 1171 results in 100 percent packet loss. With no identified customer requirements for this type of fragmentation, Red Hat has no plans to add support.

Procedure

1. Set the following value in the [ovn] section of ml2_conf.ini:

```
ovn_emit_need_to_frag = True
```

10.5.6. Configuring the native VLAN on a trunked interface

If a trunked interface or bond has a network on the native VLAN, the IP addresses are assigned directly to the bridge and there is no VLAN interface.

The following example configures a bonded interface where the External network is on the native VLAN:

```
network_config:
- type: ovs_bridge
  name: br-ex
  addresses:
  - ip_netmask: {{ external_ip }}/{{ external_cidr }}
  routes: {{ external_host_routes }}
  members:
  - type: ovs_bond
    name: bond1
    ovs_options: {{ bond_interface_ovs_options }}
    members:
    - type: interface
      name: nic3
      primary: true
    - type: interface
      name: nic4
```

NOTE

When you move the address or route statements onto the bridge, remove the corresponding VLAN interface from the bridge. Make the changes to all applicable roles. The External network is only on the controllers, so only the controller template requires a change. The Storage network is attached to all roles, so if the Storage network is on the default VLAN, all roles require modifications.

10.5.7. Increasing the maximum number of connections that netfilter tracks

The Red Hat OpenStack Platform (RHOSP) Networking service (neutron) uses netfilter connection tracking to build stateful firewalls and to provide network address translation (NAT) on virtual networks. There are some situations that can cause the kernel space to reach the maximum connection limit and result in errors such as **nf_conntrack: table full, dropping packet.** You can increase the limit for connection tracking (conntrack) and avoid these types of errors. You can increase the conntrack limit for one or more roles, or across all the nodes, in your RHOSP deployment.

Prerequisites

- A successful RHOSP undercloud installation.

Procedure

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

2. Source the undercloud credentials file:

   ```
   $ source ~/stackrc
   ```

3. Create a custom YAML environment file.

   Example

   ```
   $ vi /home/stack/templates/custom-environment.yaml
   ```

4. Your environment file must contain the keywords **parameter_defaults** and **ExtraSysctlSettings**. Enter a new value for the maximum number of connections that netfilter can track in the variable, **net.nf_conntrack_max**.

   ### Example

   In this example, you can set the conntrack limit across all hosts in your RHOSP deployment:

   ```
   parameter_defaults:
     ExtraSysctlSettings:
       net.nf_conntrack_max:
         value: 500000
   ```

   Use the **<role>Parameter** parameter to set the conntrack limit for a specific role:

   ```
   parameter_defaults:
     <role>Parameters:
       ExtraSysctlSettings:
         net.nf_conntrack_max:
           value: <simultaneous_connections>
   ```

   - Replace **<role>** with the name of the role.
     For example, use **ControllerParameters** to set the conntrack limit for the Controller role, or **ComputeParameters** to set the conntrack limit for the Compute role.

   - Replace **<simultaneous_connections>** with the quantity of simultaneous connections that you want to allow.

     ### Example

     In this example, you can set the conntrack limit for only the Controller role in your RHOSP deployment:

     ```
     parameter_defaults:
       ControllerParameters:
         ExtraSysctlSettings:
           net.nf_conntrack_max:
             value: 500000
     ```

     NOTE

     The default value for **net.nf_conntrack_max** is **500000** connections. The maximum value is: **4294967295**.

5. Run the deployment command and include the core heat templates, environment files, and this new custom environment file.

   IMPORTANT

   The order of the environment files is important as the parameters and resources defined in subsequent environment files take precedence.

   ### Example

```
$ openstack overcloud deploy --templates \
-e /home/stack/templates/custom-environment.yaml
```

Additional resources

- Environment files in the *Director Installation and Usage* guide
- Including environment files in overcloud creation in the *Director Installation and Usage* guide

10.6. NETWORK INTERFACE BONDING

You can use various bonding options in your custom network configuration.

10.6.1. Network interface bonding for overcloud nodes

You can bundle multiple physical NICs together to form a single logical channel known as a bond. You can configure bonds to provide redundancy for high availability systems or increased throughput.

Red Hat OpenStack Platform supports Open vSwitch (OVS) kernel bonds, OVS-DPDK bonds, and Linux kernel bonds.

Table 10.10. Supported interface bonding types

Bond type	Type value	Allowed bridge types	Allowed members
OVS kernel bonds	**ovs_bond**	**ovs_bridge**	**interface**
OVS-DPDK bonds	**ovs_dpdk_bond**	**ovs_user_bridge**	**ovs_dpdk_port**
Linux kernel bonds	**linux_bond**	**ovs_bridge** or **linux_bridge**	**interface**

IMPORTANT

Do not combine **ovs_bridge** and **ovs_user_bridge** on the same node.

10.6.2. Creating Open vSwitch (OVS) bonds

You create OVS bonds in your network interface templates. For example, you can create a bond as part of an OVS user space bridge:

```
- type: ovs_user_bridge
  name: br-dpdk0
  members:
  - type: ovs_dpdk_bond
    name: dpdkbond0
    rx_queue: {{ num_dpdk_interface_rx_queues }}
    members:
    - type: ovs_dpdk_port
      name: dpdk0
      members:
      - type: interface
      name: nic4
    - type: ovs_dpdk_port
      name: dpdk1
      members:
      - type: interface
        name: nic5
```

In this example, you create the bond from two DPDK ports.

The **ovs_options** parameter contains the bonding options. You can configure a bonding options in a network environment file with the **BondInterfaceOvsOptions** parameter:

```
environment_parameters:
  BondInterfaceOvsOptions: "bond_mode=active_backup"
```

10.6.3. Open vSwitch (OVS) bonding options

You can set various Open vSwitch (OVS) bonding options with the **ovs_options** heat parameter in your NIC template files.

bond_mode=balance-slb

Source load balancing (slb) balances flows based on source MAC address and output VLAN, with periodic rebalancing as traffic patterns change. When you configure a bond with the **balance-slb** bonding option, there is no configuration required on the remote switch. The Networking service (neutron) assigns each source MAC and VLAN pair to a link and transmits all packets from that MAC and VLAN through that link. A simple hashing algorithm based on source MAC address and VLAN number is used, with periodic rebalancing as traffic patterns change. The **balance-slb** mode is similar to mode 2 bonds used by the Linux bonding driver. You can use this mode to provide load balancing even when the switch is not configured to use LACP.

bond_mode=active-backup

When you configure a bond using **active-backup** bond mode, the Networking service keeps one NIC in standby. The standby NIC resumes network operations when the active connection fails. Only one MAC address is presented to the physical switch. This mode does not require switch configuration, and works when the links are connected to separate switches. This mode does not provide load balancing.

lacp=[active | passive | off]

Controls the Link Aggregation Control Protocol (LACP) behavior. Only certain switches support LACP. If your switch does not support LACP, use **bond_mode=balance-slb** or **bond_mode=active-backup**.

other-config:lacp-fallback-ab=true

Set active-backup as the bond mode if LACP fails.

other_config:lacp-time=[fast | slow]

Set the LACP heartbeat to one second (fast) or 30 seconds (slow). The default is slow.

other_config:bond-detect-mode=[miimon | carrier]

Set the link detection to use miimon heartbeats (miimon) or monitor carrier (carrier). The default is carrier.

other_config:bond-miimon-interval=100

If using miimon, set the heartbeat interval (milliseconds).

bond_updelay=1000

Set the interval (milliseconds) that a link must be up to be activated to prevent flapping.

other_config:bond-rebalance-interval=10000

Set the interval (milliseconds) that flows are rebalancing between bond members. Set this value to zero to disable flow rebalancing between bond members.

10.6.4. Using Link Aggregation Control Protocol (LACP) with Open vSwitch (OVS) bonding modes

You can use bonds with the optional Link Aggregation Control Protocol (LACP). LACP is a negotiation protocol that creates a dynamic bond for load balancing and fault tolerance.

Use the following table to understand support compatibility for OVS kernel and OVS-DPDK bonded interfaces in conjunction with LACP options.

IMPORTANT

The OVS/OVS-DPDK **balance-tcp** mode is available as a technology preview only.

IMPORTANT

On control and storage networks, Red Hat recommends that you use Linux bonds with VLAN and LACP, because OVS bonds carry the potential for control plane disruption that can occur when OVS or the neutron agent is restarted for updates, hot fixes, and other events. The Linux bond/LACP/VLAN configuration provides NIC management without the OVS disruption potential.

Table 10.11. LACP options for OVS kernel and OVS-DPDK bond modes

Objective	OVS bond mode	Compatible LACP options	Notes
High availability (active-passive)	**active-backup**	**active**, **passive**, or **off**	
Increased throughput (active-active)	**balance-slb**	**active**, **passive**, or **off**	• Performance is affected by extra parsing per packet. • There is a potential for vhost-user lock contention.

	balance-tcp	**active** or **passive**	• **Tech preview only**. Not recommended for use in production. • Recirculation needed for L4 hashing has a performance impact. • As with balance-slb, performance is affected by extra parsing per packet and there is a potential for vhost-user lock contention. • LACP must be enabled.

10.6.5. Creating Linux bonds

You create Linux bonds in your network interface templates. For example, you can create a Linux bond that bonds two interfaces:

```
- type: linux_bond
  name: bond_api
  mtu: {{ min_viable_mtu_ctlplane }}
  use_dhcp: false
  bonding_options: {{ bond_interface_ovs_options }}
  dns_servers: {{ ctlplane_dns_nameservers }}
  domain: {{ dns_search_domains }}
  members:
  - type: interface
    name: nic2
    mtu: {{ min_viable_mtu_ctlplane }}
    primary: true
  - type: interface
    name: nic3
    mtu: {{ min_viable_mtu_ctlplane }}
```

The **bonding_options** parameter sets the specific bonding options for the Linux bond.

mode

Sets the bonding mode, which in the example is **802.3ad** or LACP mode. For more information about Linux bonding modes, see "Upstream Switch Configuration Depending on the Bonding Modes" in the Red Hat Enterprise Linux 9 Configuring and Managing Networking guide.

lacp_rate

Defines whether LACP packets are sent every 1 second, or every 30 seconds.

updelay

Defines the minimum amount of time that an interface must be active before it is used for traffic. This minimum configuration helps to mitigate port flapping outages.

miimon

The interval in milliseconds that is used for monitoring the port state using the MIIMON functionality of the driver.

Use the following additional examples as guides to configure your own Linux bonds:

- Linux bond set to **active-backup** mode with one VLAN:

```
....

- type: linux_bond
  name: bond_api
  mtu: {{ min_viable_mtu_ctlplane }}
  use_dhcp: false
  bonding_options: "mode=active-backup"
  dns_servers: {{ ctlplane_dns_nameservers }}
  domain: {{ dns_search_domains }}
  members:
  - type: interface
    name: nic2
    mtu: {{ min_viable_mtu_ctlplane }}
    primary: true
  - type: interface
    name: nic3
    mtu: {{ min_viable_mtu_ctlplane }}
  - type: vlan
    mtu: {{ internal_api_mtu }}
    vlan_id: {{ internal_api_vlan_id }}
    addresses:
    - ip_netmask:
        {{ internal_api_ip }}/{{ internal_api_cidr }}
      routes:
        {{ internal_api_host_routes }}
```

- Linux bond on OVS bridge. Bond set to **802.3ad** LACP mode with one VLAN:

```
- type: linux_bond
  name: bond_tenant
  mtu: {{ min_viable_mtu_ctlplane }}
  bonding_options: "mode=802.3ad updelay=1000 miimon=100"
  use_dhcp: false
  dns_servers: {{ ctlplane_dns_nameserver }}
  domain: {{ dns_search_domains }}
  members:
    - type: interface
      name: p1p1
      mtu: {{ min_viable_mtu_ctlplane }}
    - type: interface
      name: p1p2
      mtu: {{ min_viable_mtu_ctlplane }}
    - type: vlan
      mtu: {{ tenant_mtu }}
```

```
vlan_id: {{ tenant_vlan_id }}
addresses:
 - ip_netmask:
    {{ tenant_ip }}/{{ tenant_cidr }}
routes:
 {{ tenant_host_routes }}
```

IMPORTANT

You must set up **min_viable_mtu_ctlplane** before you can use it. Copy **/usr/share/ansible/roles/tripleo_network_config/templates/2_linux_bonds_vlans.j2** to your templates directory and modify it for your needs. For more information, see Adding a composable network , and refer to the steps that pertain to the network configuration template.

10.7. UPDATING THE FORMAT OF YOUR NETWORK CONFIGURATION FILES

The format of the network configuration **yaml** files has changed in Red Hat OpenStack Platform (RHOSP) 17.0. The structure of the network configuration file **network_data.yaml** has changed, and the NIC template file format has changed from **yaml** file format to Jinja2 ansible format, **j2**.

You can convert your existing network configuration file in your current deployment to the RHOSP 17+ format by using the following conversion tools:

- **convert_v1_net_data.py**
- **convert_heat_nic_config_to_ansible_j2.py**

You can also manually convert your existing NIC template files.

The files you need to convert include the following:

- **network_data.yaml**
- Controller NIC templates
- Compute NIC templates
- Any other custom network files

10.7.1. Updating the format of your network configuration file

The format of the network configuration **yaml** file has changed in Red Hat OpenStack Platform (RHOSP) 17.0. You can convert your existing network configuration file in your current deployment to the RHOSP 17+ format by using the **convert_v1_net_data.py** conversion tool.

Procedure

1. Download the conversion tool:
   - **/usr/share/openstack-tripleo-heat-templates/tools/convert_v1_net_data.py**
2. Convert your RHOSP 16+ network configuration file to the RHOSP 17+ format:

```
$ python3 convert_v1_net_data.py <network_config>.yaml
```

- Replace **<network_config>** with the name of the existing configuration file that you want to convert, for example, **network_data.yaml**.

10.7.2. Automatically converting NIC templates to Jinja2 Ansible format

The NIC template file format has changed from **yaml** file format to Jinja2 Ansible format, **j2**, in Red Hat OpenStack Platform (RHOSP) 17.0.

You can convert your existing NIC template files in your current deployment to the Jinja2 format by using the **convert_heat_nic_config_to_ansible_j2.py** conversion tool.

You can also manually convert your existing NIC template files. For more information, see Manually converting NIC templates to Jinja2 Ansible format.

The files you need to convert include the following:

- Controller NIC templates
- Compute NIC templates
- Any other custom network files

Procedure

1. Download the conversion tool:
    - **/usr/share/openstack-tripleo-heat-templates/tools/convert_heat_nic_config_to_ansible_j2.py**
2. Convert your Compute and Controller NIC tempate files, and any other custom network files, to the Jinja2 Ansible format:

```
$ python3 convert_heat_nic_config_to_ansible_j2.py \
 [--stack <overcloud> | --standalone] --networks_file <network_config.yaml> \
 <network_template>.yaml
```

- Replace **<overcloud>** with the name or UUID of the overcloud stack. If **--stack** is not specified, the stack defaults to **overcloud**.

NOTE

You can use the **--stack** option only on your RHOSP 16 deployment because it requires the Orchestration service (heat) to be running on the undercloud node. Starting with RHOSP 17, RHOSP deployments use ephemeral heat, which runs the Orchestration service in a container. If the Orchestration service is not available, or you have no stack, then use the **--standalone** option instead of **--stack**.

- Replace **<network_config.yaml>** with the name of the configuration file that describes the network deployment, for example, **network_data.yaml**.

- Replace **<network_template>** with the name of the network configuration file you want to convert.

Repeat this command until you have converted all your custom network configuration files. The **convert_heat_nic_config_to_ansible_j2.py** script generates a **.j2** file for each **yaml** file you pass to it for conversion.

3. Inspect each generated **.j2** file to ensure the configuration is correct and complete for your environment, and manually address any comments generated by the tool that highlight where the configuration could not be converted. For more information about manually converting the NIC configuration to Jinja2 format, see Heat parameter to Ansible variable mappings .

4. Configure the ***NetworkConfigTemplate** parameters in your **network-environment.yaml** file to point to the generated **.j2** files:

```
parameter_defaults:
  ControllerNetworkConfigTemplate: '/home/stack/templates/custom-nics/controller.j2'
  ComputeNetworkConfigTemplate: '/home/stack/templates/custom-nics/compute.j2'
```

5. Delete the **resource_registry** mappings from your **network-environment.yaml** file for the old network configuration files:

```
resource_registry:
  OS::TripleO::Compute::Net::SoftwareConfig: /home/stack/templates/nic-
configs/compute.yaml
  OS::TripleO::Controller::Net::SoftwareConfig: /home/stack/templates/nic-
configs/controller.yaml
```

10.7.3. Manually converting NIC templates to Jinja2 Ansible format

The NIC template file format has changed from **yaml** file format to Jinja2 Ansible format, **j2**, in Red Hat OpenStack Platform (RHOSP) 17.0.

You can manually convert your existing NIC template files.

You can also convert your existing NIC template files in your current deployment to the Jinja2 format by using the **convert_heat_nic_config_to_ansible_j2.py** conversion tool. For more information, see Automatically converting NIC templates to Jinja2 ansible format .

The files you need to convert include the following:

- Controller NIC templates
- Compute NIC templates
- Any other custom network files

Procedure

1. Create a Jinja2 template. You can create a new template by using the **os-net-config** schema, or copy and edit an example template from the **/usr/share/ansible/roles/tripleo_network_config/templates/** directory on the undercloud node.

2. Replace the heat intrinsic functions with Jinja2 filters. For example, use the following filter to calculate the **min_viable_mtu**:

```
{% set mtu_list = [ctlplane_mtu] %}
{% for network in role_networks %}
{{ mtu_list.append(lookup('vars', networks_lower[network] ~ '_mtu')) }}
{%- endfor %}
{% set min_viable_mtu = mtu_list | max %}
```

3. Use Ansible variables to configure the network properties for your deployment. You can configure each individual network manually, or programatically configure each network by iterating over **role_networks**:

   - To manually configure each network, replace each **get_param** function with the equivalent Ansible variable. For example, if your current deployment configures **vlan_id** by using **get_param: InternalApiNetworkVlanID**, then add the following configuration to your template:

     ```
     vlan_id: {{ internal_api_vlan_id }}
     ```

     Table 10.12. Example network property mapping from heat parameters to Ansible **vars**

yaml file format	Jinja2 ansible format, **j2**

```
- type: vlan
  device: nic2
  vlan_id:
    get_param:
InternalApiNetworkVlanID
  addresses:
  - ip_netmask:
      get_param: InternalApiIpSubnet
```
	```
	- type: vlan
	  device: nic2
	  vlan_id: {{ internal_api_vlan_id }}
	  addresses:
	  - ip_netmask: {{ internal_api_ip }}/{{
	internal_api_cidr }}
	```

   - To programatically configure each network, add a Jinja2 for-loop structure to your template that retrieves the available networks by their role name by using **role_networks**.

     Example

     ```
     {% for network in role_networks %}
       - type: vlan
         mtu: {{ lookup('vars', networks_lower[network] ~ '_mtu') }}
         vlan_id: {{ lookup('vars', networks_lower[network] ~ '_vlan_id') }}
         addresses:
         - ip_netmask: {{ lookup('vars', networks_lower[network] ~ '_ip') }}/{{ lookup('vars',
     networks_lower[network] ~ '_cidr') }}
         routes: {{ lookup('vars', networks_lower[network] ~ '_host_routes') }}
     {%- endfor %}
     ```

   For a full list of the mappings from the heat parameter to the Ansible **vars** equivalent, see Heat parameter to Ansible variable mappings.

4. Configure the ***NetworkConfigTemplate** parameters in your **network-environment.yaml** file to point to the generated **.j2** files:

```
parameter_defaults:
  ControllerNetworkConfigTemplate: '/home/stack/templates/custom-nics/controller.j2'
  ComputeNetworkConfigTemplate: '/home/stack/templates/custom-nics/compute.j2'
```

5. Delete the **resource_registry** mappings from your **network-environment.yaml** file for the old network configuration files:

```
resource_registry:
  OS::TripleO::Compute::Net::SoftwareConfig: /home/stack/templates/nic-
configs/compute.yaml
  OS::TripleO::Controller::Net::SoftwareConfig: /home/stack/templates/nic-
configs/controller.yaml
```

10.7.4. Heat parameter to Ansible variable mappings

The NIC template file format has changed from **yaml** file format to Jinja2 ansible format, **j2**, in Red Hat OpenStack Platform (RHOSP) 17.x.

To manually convert your existing NIC template files to Jinja2 ansible format, you can map your heat parameters to Ansible variables to configure the network properties for pre-provisioned nodes in your deployment. You can also map your heat parameters to Ansible variables if you run **openstack overcloud node provision** without specifying the **--network-config** optional argument.

For example, if your current deployment configures **vlan_id** by using **get_param: InternalApiNetworkVlanID**, then replace it with the following configuration in your new Jinja2 template:

```
vlan_id: {{ internal_api_vlan_id }}
```

NOTE

If you provision your nodes by running **openstack overcloud node provision** with the **--network-config** optional argument, you must configure the network properties for your deploying by using the parameters in **overcloud-baremetal-deploy.yaml**. For more information, see Heat parameter to provisioning definition file mappings .

The following table lists the available mappings from the heat parameter to the Ansible **vars** equivalent.

Table 10.13. Mappings from heat parameters to Ansible**vars**

Heat parameter	Ansible **vars**
BondInterfaceOvsOptions	**{{ bond_interface_ovs_options }}**
ControlPlaneIp	**{{ ctlplane_ip }}**
ControlPlaneDefaultRoute	**{{ ctlplane_gateway_ip }}**
ControlPlaneMtu	**{{ ctlplane_mtu }}**
ControlPlaneStaticRoutes	**{{ ctlplane_host_routes }}**

Heat parameter	Ansible **vars**
ControlPlaneSubnetCidr	**{{ ctlplane_subnet_cidr }}**
DnsSearchDomains	**{{ dns_search_domains }}**
DnsServers	**{{ ctlplane_dns_nameservers }}** NOTE This Ansible variable is populated with the IP address configured in **undercloud.conf** for **DEFAULT/undercloud_nameservers** and **%SUBNET_SECTION%/dns_nameservers**. The configuration of **%SUBNET_SECTION%/dns_nameservers** overrides the configuration of **DEFAULT/undercloud_nameservers**, so that you can use different DNS servers for the undercloud and the overcloud, and different DNS servers for nodes on different provisioning subnets.
NumDpdkInterfaceRxQueues	**{{ num_dpdk_interface_rx_queues }}**

Configuring a heat parameter that is not listed in the table

To configure a heat parameter that is not listed in the table, you must configure the parameter as a **{{role.name}}ExtraGroupVars**. After you have configured the parameter as a **{{role.name}}ExtraGroupVars** parameter, you can then use it in your new template. For example, to configure the **StorageSupernet** parameter, add the following configuration to your network configuration file:

```
parameter_defaults:
  ControllerExtraGroupVars:
    storage_supernet: 172.16.0.0/16
```

You can then add **{{ storage_supernet }}** to your Jinja2 template.

WARNING

This process will not work if the **--network-config** option is used with node provisioning. Users requiring custom vars should not use the **--network-config** option. Instead, after creating the Heat stack, apply the node network configuration to the **config-download** ansible run.

Converting the Ansible variable syntax to programmatically configure each network

When you use a Jinja2 for-loop structure to retrieve the available networks by their role name by iterating over **role_networks**, you need to retrieve the lower case name for each network role to prepend to each property. Use the following structure to convert the Ansible **vars** from the above table to the required syntax:

{{ lookup('vars', networks_lower[network] ~ '_<property>') }}

- Replace **<property>** with the property that you are setting, for example, **ip**, **vlan_id**, or **mtu**.

For example, to populate the value for each **NetworkVlanID** dynamically, replace **{{ <network_name>_vlan_id }}** with the following configuration:

```
{{ lookup('vars', networks_lower[network] ~ '_vlan_id') }}`
```

10.7.5. Heat parameter to provisioning definition file mappings

If you provision your nodes by running the **openstack overcloud node provision** command with the **--network-config** optional argument, you must configure the network properties for your deployment by using the parameters in the node definition file **overcloud-baremetal-deploy.yaml**.

If your deployment uses pre-provisioned nodes, you can map your heat parameters to Ansible variables to configure the network properties. You can also map your heat parameters to Ansible variables if you run **openstack overcloud node provision** without specifying the **--network-config** optional argument. For more information about configuring network properties by using Ansible variables, see Heat parameter to Ansible variable mappings.

The following table lists the available mappings from the heat parameter to the **network_config** property equivalent in the node definition file **overcloud-baremetal-deploy.yaml**.

Table 10.14. Mappings from heat parameters to node definition file**overcloud-baremetal-deploy.yaml**

Heat parameter	**network_config** property
BondInterfaceOvsOptions	**bond_interface_ovs_options**
DnsSearchDomains	**dns_search_domains**
NetConfigDataLookup	**net_config_data_lookup**
NeutronPhysicalBridge	**physical_bridge_name**
NeutronPublicInterface	**public_interface_name**
NumDpdkInterfaceRxQueues	**num_dpdk_interface_rx_queues**
{{role.name}}NetworkConfigUpdate	**network_config_update**

The following table lists the available mappings from the heat parameter to the property equivalent in the networks definition file **network_data.yaml**.

Table 10.15. Mappings from heat parameters to networks definition file**network_data.yaml**

Heat parameter	IPv4 **network_data.yaml** property	IPv6 **network_data.yaml** property

<network_name>IpSubnet	`- name: <network_name>`
	`  subnets:`
	`    subnet01:`
	`      ip_subnet: 172.16.1.0/24`
		`- name: <network_name>`
		`  subnets:`
		`    subnet01:`
		`      ipv6_subnet:`
		`2001:db8:a::/64`

<network_name>NetworkVlanID	`- name: <network_name>`
	`  subnets:`
	`    subnet01:`
	`      ...`
	`      vlan: <vlan_id>`
		`- name: <network_name>`
		`  subnets:`
		`    subnet01:`
		`      ...`
		`      vlan: <vlan_id>`

<network_name>Mtu	`- name: <network_name>`
	`  mtu:`
		`- name: <network_name>`
		`  mtu:`

<network_name>InterfaceDefaultRoute	`- name: <network_name>`
	`  subnets:`
	`    subnet01:`
	`      ip_subnet: 172.16.16.0/24`
	`      gateway_ip: 172.16.16.1`
		`- name: <network_name>`
		`  subnets:`
		`    subnet01:`
		`      ipv6_subnet:`
		`2001:db8:a::/64`
		`      gateway_ipv6:`
		`2001:db8:a::1`

<network_name>InterfaceRoutes	`- name: <network_name>`
	`  subnets:`
	`    subnet01:`
	`      ...`
	`      routes:`
	`        - destination:`
	`172.18.0.0/24`
	`          nexthop: 172.18.1.254`
		`- name: <network_name>`
		`  subnets:`
		`    subnet01:`
		`      ...`
		`      routes_ipv6:`
		`        - destination:`
		`2001:db8:b::/64`
		`          nexthop: 2001:db8:a::1`

10.7.6. Changes to the network data schema

The network data schema was updated in Red Hat OpenStack Platform (RHOSP) 17. The main differences between the network data schema used in RHOSP 16 and earlier, and network data schema used in RHOSP 17 and later, are as follows:

- The base subnet has been moved to the **subnets** map. This aligns the configuration for non-routed and routed deployments, such as spine-leaf networking.

- The **enabled** option is no longer used to ignore disabled networks. Instead, you must remove disabled networks from the configuration file.
- The **compat_name** option is no longer required as the heat resource that used it has been removed.
- The following parameters are no longer valid at the network level: **ip_subnet**, **gateway_ip**, **allocation_pools**, **routes**, **ipv6_subnet**, **gateway_ipv6**, **ipv6_allocation_pools**, and **routes_ipv6**. These parameters are still used at the subnet level.
- A new parameter, **physical_network**, has been introduced, that is used to create ironic ports in **metalsmith**.
- New parameters **network_type** and **segmentation_id** replace **{{network.name}}NetValueSpecs** used to set the network type to **vlan**.
- The following parameters have been deprecated in RHOSP 17:
  - **{{network.name}}NetCidr**
  - **{{network.name}}SubnetName**
  - **{{network.name}}Network**
  - **{{network.name}}AllocationPools**
  - **{{network.name}}Routes**
  - **{{network.name}}SubnetCidr_{{subnet}}**
  - **{{network.name}}AllocationPools_{{subnet}}**
  - **{{network.name}}Routes_{{subnet}}**

CHAPTER 11. PROVISIONING AND DEPLOYING YOUR OVERCLOUD

To create an overcloud you must perform the following tasks:

1. Provision the network resources for your physical networks:
   a. If you are deploying network isolation or a custom composable network, then create a network definition file in YAML format.
   b. Run the network provisioning command, including the network definition file.
   c. Create a network Virtual IP (VIP) definition file in YAML format.
   d. Run the network VIP provisioning command, including the network VIP definition file.
2. Provision your bare metal nodes:
   a. Create a node definition file in YAML format.
   b. Run the bare-metal node provisioning command, including the node definition file.
3. Deploy your overcloud.
   a. Run the deployment command, including the heat environment files that the provisioning commands generate.

11.1. PROVISIONING THE OVERCLOUD NETWORKS

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

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

11.1.1. Configuring and provisioning overcloud network definitions

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

Prerequisites

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

Procedure

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

```
$ source ~/stackrc
```

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

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

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

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

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

5. Provision the overcloud networks:

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

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

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

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

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

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

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

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

Next steps

- Configuring and provisioning network VIPs for the overcloud

11.1.2. Configuring and provisioning network VIPs for the overcloud

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

Prerequisites

- The undercloud is installed. For more information, see Installing director.
- Your overcloud networks are provisioned. For more information, see Configuring and provisioning overcloud network definitions.

Procedure

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

   ```
   $ source ~/stackrc
   ```

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

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

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

   ```
   - network: external
     dns_name: overcloud
   - network: ctlplane
     dns_name: overcloud
   ```

4. Configure any other network VIP attributes for your environment. For more information about the properties you can use to configure VIP attributes in your VIP definition file, see Adding a composable network.

5. Provision the network VIPs:

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

   - Optional: Include the **--templates** option to use your own templates instead of the default templates located in **/usr/share/openstack-tripleo-heat-templates**. Replace **<templates_directory>** with the path to the directory that contains your templates.
   - Replace **<stack>** with the name of the stack for which the network VIPs are provisioned, for example, **overcloud**.

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

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

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

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

Next steps

- Provisioning bare metal overcloud nodes

11.2. PROVISIONING BARE METAL OVERCLOUD NODES

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

1. Register the bare-metal nodes for your overcloud.
2. Provide director with an inventory of the hardware of the bare-metal nodes.
3. Configure the quantity, attributes, and network layout of the bare-metal nodes in a node definition file.
4. Assign each bare metal node a resource class that matches the node to its designated role.

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

11.2.1. Registering nodes for the overcloud

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

Procedure

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

Example JSON template

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

Example YAML template

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

This template contains the following attributes:

name

The logical name for the node.

ports

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

- **address**: The MAC address for the network interface on the node. Use only the MAC address for the Provisioning NIC of each system.
- **physical_network**: The physical network that is connected to the Provisioning NIC.
- **local_link_connection**: If you use IPv6 provisioning and LLDP does not correctly populate the local link connection during introspection, you must include fake data with the **switch_id** and **port_id** fields in the **local_link_connection** parameter. For more information on how to include fake data, see Using director introspection to collect bare metal node hardware information.

cpu

(Optional) The number of CPUs on the node.

memory

(Optional) The amount of memory in MB.

disk

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

arch

(Optional) The system architecture.

pm_type

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

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

pm_user; pm_password

The IPMI username and password.

pm_addr

The IP address of the IPMI device.

2. Verify the template formatting and syntax:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

TIP

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

Prerequisites

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

Procedure

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

   ```
   $ source ~/stackrc
   ```

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

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

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

   NOTE

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

4. Review the results of the validation report.
5. Optional: Review detailed output from a specific validation:

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

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

   IMPORTANT

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

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

   - Inspect the hardware attributes of all nodes:

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

- Use the **--all-manageable** option to introspect only the nodes that are in a managed state. In this example, all nodes are in a managed state.
- Use the **--provide** option to reset all nodes to an **available** state after introspection.

- Inspect the hardware attributes of specific nodes:

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

  - Use the **--provide** option to reset all the specified nodes to an **available** state after introspection.
  - Replace **<node1>**, **[node2]**, and all nodes up to **[noden]** with the UUID of each node that you want to introspect.

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

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

IMPORTANT

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

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

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

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

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

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

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

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

11.2.2.2. Manually configuring bare-metal node hardware information

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

TIP

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

Prerequisites

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

Procedure

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

   ```
   $ source ~/stackrc
   ```

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

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

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

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

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

   - Replace **<node>** with the ID of the bare metal node.
   - Replace **<cpu>** with the number of CPUs.
   - Replace **<ram>** with the RAM in MB.
   - Replace **<disk>** with the disk size in GB.
   - Replace **<arch>** with the architecture type.

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

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

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

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

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

- Replace **<node>** with the ID of the bare metal node.
- Replace **<property>** and **<value>** with details about the disk that you want to use for deployment, for example **root_device='{"size": "128"}'**
  RHOSP supports the following properties:
  - **model** (String): Device identifier.
  - **vendor** (String): Device vendor.
  - **serial** (String): Disk serial number.
  - **hctl** (String): Host:Channel:Target:Lun for SCSI.
  - **size** (Integer): Size of the device in GB.
  - **wwn** (String): Unique storage identifier.
  - **wwn_with_extension** (String): Unique storage identifier with the vendor extension appended.
  - **wwn_vendor_extension** (String): Unique vendor storage identifier.
  - **rotational** (Boolean): True for a rotational device (HDD), otherwise false (SSD).
  - **name** (String): The name of the device, for example: /dev/sdb1 Use this property only for devices with persistent names.

NOTE

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

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

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

- Replace **<node_uuid>** with the unique ID of the bare metal node.
- Replace **<mac_address>** with the MAC address of the NIC used to PXE boot.

8. Validate the configuration of the node:

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

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

- **False**: The interface has failed validation. If the reason provided includes missing the **instance_info.image_source** parameter, this might be because it is populated during provisioning, therefore it has not been set at this point. If you are using a whole disk image, then you might need to only set **image_source** to pass the validation.
- **True**: The interface has passed validation.
- **None**: The interface is not supported for your driver.

11.2.3. Provisioning bare metal nodes for the overcloud

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

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

Prerequisites

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

Procedure

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

```
$ source ~/stackrc
```

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

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

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

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

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

Specific nodes

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

All nodes for a role

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

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

Specific nodes

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

```
- name: Controller
  count: 3
  defaults:
    network_config:
      template: /home/stack/templates/nic-config/myController.j2
      default_route_network:
      - external
    instances:
      - hostname: overcloud-controller-0
        name: node00
        networks:
          - network: ctlplane
            vif: true
          - network: external
            subnet: external_subnet
          - network: internal_api
            subnet: internal_api_subnet01
            fixed_ip: 172.21.11.100
          - network: storage
            subnet: storage_subnet01
          - network: storage_mgmt
            subnet: storage_mgmt_subnet01
          - network: tenant
            subnet: tenant_subnet01
```

All nodes for a role

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

```
- name: Controller
  count: 3
  defaults:
    networks:
    - network: ctlplane
      vif: true
    - network: external
      subnet: external_subnet
    - network: internal_api
      subnet: internal_api_subnet01
    - network: storage
      subnet: storage_subnet01
    - network: storage_mgmt
      subnet: storage_mgmt_subnet01
    - network: tenant
      subnet: tenant_subnet01
    network_config:
      template: /home/stack/templates/nic-config/myController.j2 1
      default_route_network:
      - external
- name: Compute
  count: 3
  defaults:
    networks:
    - network: ctlplane
      vif: true
    - network: internal_api
      subnet: internal_api_subnet02
    - network: tenant
      subnet: tenant_subnet02
    - network: storage
      subnet: storage_subnet02
    network_config:
      template: /home/stack/templates/nic-config/myCompute.j2
```

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

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

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

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

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

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

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

10. Provision the overcloud nodes:

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

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

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

- Include the **--network-config** optional argument to provide the network definitions to the **cli-overcloud-node-network-config.yaml** Ansible playbook.

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

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

11. Monitor the provisioning progress in a separate terminal:

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

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

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

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

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

13. Verify the association of nodes to hostnames:

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

Next steps

- Configuring and deploying the overcloud

11.2.4. Bare-metal node provisioning attributes

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

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

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

Table 11.1. Role properties

Property	Value
name	(Mandatory) Role name.
count	The number of nodes that you want to provision for this role. The default value is **1**.
defaults	A dictionary of default values for **instances** entry properties. An **instances** entry property overrides any defaults that you specify in the **defaults** parameter.
instances	A dictionary of values that you can use to specify attributes for specific nodes. For more information about supported properties in the **instances** parameter, see **defaults** and **instances** properties. The number of nodes defined must not be greater than the value of the **count** parameter.
hostname_format	Overrides the default hostname format for this role. The default generated hostname is derived from the overcloud stack name, the role, and an incrementing index, all in lower case. For example, the default format for the Controller role is **%stackname%-controller-%index%**. Only the Compute role does not follow the role name rule. The Compute default format is **%stackname%-novacompute-%index%**.

Property	Value
ansible_playbooks	A dictionary of values for Ansible playbooks and Ansible variables. The playbooks are run against the role instances after node provisioning, prior to the node network configuration. For more information about specifying Ansible playbooks, see **ansible_playbooks** properties.

Table 11.2. **defaults** and **instances** properties

Property	Value
hostname	(**instances** only) Specifies the hostname of the node that the**instance** properties apply to. The hostname is derived from the **hostname_format** property. You can use custom hostnames.
name	(**instances** only) The name of the node that you want to provision.
image	Details of the image that you want to provision onto the node. For information about supported **image** properties, see **image** properties.
capabilities	Selection criteria to match the node capabilities.
config_drive	Add data and first-boot commands to the config-drive passed to the node. For more information, see **config_drive** properties. NOTE Only use **config_drive** for configuration that must be performed on first boot. For all other custom configurations, create an Ansible playbook and use the **ansible_playbooks** property to execute the playbook against the role instances after node provisioning.
managed	Set to **true** (default) to provision the instance with metalsmith. Set to**false** to handle the instance as pre-provisioned.
networks	List of dictionaries that represent instance networks. For more information about configuring network attributes, see **network** properties.
network_config	Link to the network configuration file for the role or instance. For more information about configuring the link to the network configuration file, see **network_config** properties.
profile	Selection criteria to for profile matching. For more information, see Designating overcloud nodes for roles by matching profiles
provisioned	Set to **true** (default) to provision the node. Set to **false** to unprovision a node. For more information, see Scaling down bare-metal nodes

Property	Value
resource_class	Selection criteria to match the resource class of the node. The default value is **baremetal**. For more information, seeDesignating overcloud nodes for roles by matching resource classes.
root_size_gb	Size of the root partition in GiB. The default value is **49**.
swap_size_mb	Size of the swap partition in MiB.
traits	A list of traits as selection criteria to match the node traits.

Table 11.3. **image** properties

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

Table 11.4. **network** properties

Property	Value
fixed_ip	The specific IP address that you want to use for this network.
network	The network where you want to create the network port.
subnet	The subnet where you want to create the network port.

Property	Value
port	Existing port to use instead of creating a new port.
vif	Set to **true** on the provisioning network (**ctlplane**) to attach the network as a virtual interface (VIF). Set to **false** to create the Networking service API resource without a VIF attachment.

Table 11.5. **network_config** properties

Property	Value
template	Specifies the Ansible J2 NIC configuration template to use when applying node network configuration. For information on configuring the NIC template, see Configuring overcloud networking.
physical_bridge_name	The name of the OVS bridge to create for accessing external networks. The default bridge name is **br-ex**.
public_interface_name	Specifies the name of the interface to add to the public bridge. The default interface is **nic1**.
network_config_update	Set to **true** to apply network configuration changes on update. Disabled by default.
net_config_data_lookup	Specifies the NIC mapping configuration, **os-net-config**, for each node or node group.
default_route_network	The network to use for the default route. The default route network is **ctlplane**
networks_skip_config	List of networks to skip when configuring the node networking.
dns_search_domains	A list of DNS search domains to be added to **resolv.conf**, in order of priority.
bond_interface_ovs_options	The OVS options or bonding options to use for the bond interface, for example, **lacp=active** and **bond_mode=balance-slb** for OVS bonds, and **mode=4** for Linux bonds.
num_dpdk_interface_rx_queues	Specifies the number of required RX queues for DPDK bonds or DPDK ports.

Table 11.6. **config_drive** properties

Property	Value

Property	Value

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

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

Table 11.7. **ansible_playbooks** properties

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

Property	Value

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

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

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

Prerequisites

- The undercloud is installed. For more information, see Installing director.
- The bare-metal node provisioning failed because of a node hardware failure.

Procedure

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

   ```
   $ source ~/stackrc
   ```

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

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

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

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

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

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

6. Unprovision the bare-metal nodes:

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

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

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

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

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

11.2.6. Designating overcloud nodes for roles by matching resource classes

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

NOTE

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

Prerequisites

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

Procedure

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

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

   - Replace **<resource_class>** with a name for the resource class, for example, **baremetal.CPU-PINNING**.
   - Replace **<node>** with the ID of the bare metal node.

2. Add the role to your **overcloud-baremetal-deploy.yaml** file, if not already defined.
3. Specify the resource class that you want to assign to the nodes for the role:

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

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

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

11.2.7. Designating overcloud nodes for roles by matching profiles

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

TIP

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

NOTE

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

Prerequisites

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

Procedure

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

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

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

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

   - Replace **<node>** with the name or ID of the bare metal node.
   - Replace **<profile>** with the name of the profile that matches the role profile.
   - Replace **<capability_1>**, and all capabilities up to **<capability_n>**, with each capability that you retrieved in step 1.

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

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

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

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

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

11.2.8. Configuring whole disk partitions for the Object Storage service

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

Prerequisites

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

Procedure

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

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

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

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

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

```
ObjectStorage:
  /=8GB
  /tmp=1GB
  /var/log=10GB
  /var/log/audit=2GB
  /home=1GB
  /var=10GB
  /srv=100%
```

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

11.2.9. Example node definition file

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

```
- name: Controller
  count: 3
  defaults:
    image:
      href: file:///var/lib/ironic/images/overcloud-custom.qcow2
    networks:
    - network: ctlplane
      vif: true
    - network: external
      subnet: external_subnet
    - network: internal_api
      subnet: internal_api_subnet01
    - network: storage
      subnet: storage_subnet01
    - network: storagemgmt
      subnet: storage_mgmt_subnet01
    - network: tenant
      subnet: tenant_subnet01
    network_config:
        template: /home/stack/templates/nic-config/myController.j2
        default_route_network:
        - external
    profile: nodeCapability
  instances:
  - hostname: overcloud-controller-0
    name: node00
  - hostname: overcloud-controller-1
    name: node01
  - hostname: overcloud-controller-2
    name: node02
- name: Compute
  count: 3
  defaults:
    networks:
    - network: ctlplane
      vif: true
    - network: internal_api
      subnet: internal_api_subnet02
    - network: tenant
      subnet: tenant_subnet02
    - network: storage
      subnet: storage_subnet02
    network_config:
      template: /home/stack/templates/nic-config/myCompute.j2
    resource_class: baremetal.COMPUTE
  instances:
  - hostname: overcloud-novacompute-0
    name: node04
  - hostname: overcloud-novacompute-1
    name: node05
  - hostname: overcloud-novacompute-2
    name: node06
```

11.2.10. Enabling virtual media boot

IMPORTANT

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

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

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

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

Prerequisites

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

Procedure

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

```
$ source ~/stackrc
```

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

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

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

4. Define the ESP image:

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

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

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

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

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

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

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

Hardware properties are used to identify the root disk. For more information about properties you can use to identify the root disk, see Section 11.2.12, "Properties that identify the root disk" .

Procedure

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

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

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

```
[
{
  "size": 299439751168,
  "rotational": true,
  "vendor": "DELL",
  "name": "/dev/sda",
  "wwn_vendor_extension": "0x1ea4dcc412a9632b",
  "wwn_with_extension": "0x61866da04f3807001ea4dcc412a9632b",
  "model": "PERC H330 Mini",
  "wwn": "0x61866da04f380700",
  "serial": "61866da04f3807001ea4dcc412a9632b"
}
{
  "size": 299439751168,
  "rotational": true,
  "vendor": "DELL",
  "name": "/dev/sdb",
  "wwn_vendor_extension": "0x1ea4e13c12e36ad6",
  "wwn_with_extension": "0x61866da04f380d001ea4e13c12e36ad6",
  "model": "PERC H330 Mini",
  "wwn": "0x61866da04f380d00",
  "serial": "61866da04f380d001ea4e13c12e36ad6"
}
{
  "size": 299439751168,
  "rotational": true,
  "vendor": "DELL",
  "name": "/dev/sdc",
  "wwn_vendor_extension": "0x1ea4e31e121cfb45",
  "wwn_with_extension": "0x61866da04f37fc001ea4e31e121cfb45",
  "model": "PERC H330 Mini",
  "wwn": "0x61866da04f37fc00",
  "serial": "61866da04f37fc001ea4e31e121cfb45"
}
]
```

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

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

   NOTE

   A unique hardware property is any property from the hardware introspection step that uniquely identifies the disk. For example, the following command uses the disk serial number to set the root disk:

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

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

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

11.2.12. Properties that identify the root disk

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

- **model** (String): Device identifier.
- **vendor** (String): Device vendor.
- **serial** (String): Disk serial number.
- **hctl** (String): Host:Channel:Target:Lun for SCSI.
- **size** (Integer): Size of the device in GB.
- **wwn** (String): Unique storage identifier.
- **wwn_with_extension** (String): Unique storage identifier with the vendor extension appended.
- **wwn_vendor_extension** (String): Unique vendor storage identifier.
- **rotational** (Boolean): True for a rotational device (HDD), otherwise false (SSD).
- **name** (String): The name of the device, for example: /dev/sdb1.

IMPORTANT

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

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

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

NOTE

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

Procedure

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

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

   Specific nodes

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

   All nodes for a role

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

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

   ```
   rhsm_enforce: False
   ```

4. Run the provisioning command:

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

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

11.3. CONFIGURING AND DEPLOYING THE OVERCLOUD

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

IMPORTANT

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

11.3.1. Prerequisites

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

11.3.2. Configuring your overcloud by using environment files

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

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

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

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

1. Modify parameters in the custom environment files and heat templates.
2. Run the **openstack overcloud deploy** command again with the same environment files.

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

NOTE

Open Virtual Networking (OVN) is the default networking mechanism driver in Red Hat OpenStack Platform 17.0. 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.

11.3.3. Creating an environment file for undercloud CA trust

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

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

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

- **User-provided certificates** - This definition applies when you have provided your own certificate. This can be from your own CA, or it can be self-signed. This is passed using the **undercloud_service_certificate** option. In this case, you must either trust the self-signed certificate, or the CA (depending on your deployment).
- **Auto-generated certificates** - This definition applies when you use **certmonger** to generate the certificate using its own local CA. Enable auto-generated certificates with the **generate_service_certificate** option in the **undercloud.conf** file. In this case, director generates a CA certificate at **/etc/pki/ca-trust/source/anchors/cm-local-ca.pem** and the director configures the undercloud's HAProxy instance to use a server certificate. Add the CA certificate to the **inject-trust-anchor-hiera.yaml** file to present the certificate to OpenStack Platform.

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

Procedure

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

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

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

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

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

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

BAYTAlVTMQswCQYDVQQIDAJOQzEQMA4GA1UEBwwHUmFsZWlnaDEQMA4GA1UECg
wH
```

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

NOTE

- The certificate string must follow the PEM format.
- The **CAMap** parameter might contain other certificates relevant to SSL/TLS configuration.

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

11.3.4. Disabling TSX on new deployments

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

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

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

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

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

Additional resources

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

11.3.5. Validating your overcloud configuration

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

IMPORTANT

- As a result of a change to the API in 17.0, the following validations are currently unstable:
  - switch-vlans
  - network-environment
  - dhcp-provisioning
- A **FAILED** validation does not prevent you from deploying or running Red Hat OpenStack Platform. However, a **FAILED** validation can indicate a potential issue with a production environment.

Procedure

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

```
$ source ~/stackrc
```

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

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

4. Validate your overcloud stack:

```
$ validation run \
  --group pre-deployment \
  --inventory <inventory_file>
```

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

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

5. Review the results of the validation report:

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

   - Optional: Use the **--full** option to view detailed output from the validation run.
   - Optional: Use the **--validation-log-dir** option to write the output from the validation run to the validation logs.
   - Replace **<uuid>** with the UUID of the validation run.

11.3.6. Creating your overcloud

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

Procedure

1. Collate the environment files and configuration files that you need for your overcloud environment, both the unedited heat template files provided with your director installation, and the custom environment files you created. This should include the following files:
   - **overcloud-baremetal-deployed.yaml** node definition file.

- **overcloud-networks-deployed.yaml** network definition file.
- **overcloud-vip-deployed.yaml** network VIP definition file.
- The location of the container images for containerized OpenStack services.
- Any environment files for Red Hat CDN or Satellite registration.
- Any other custom environment files.

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

3. Construct your **openstack overcloud deploy** command, specifying the configuration files and templates in the required order, for example:

```
(undercloud) $ openstack overcloud deploy --templates \
 [-n /home/stack/templates/network_data.yaml \ ]
  -e /home/stack/templates/overcloud-baremetal-deployed.yaml\
  -e /home/stack/templates/overcloud-networks-deployed.yaml\
  -e /home/stack/templates/overcloud-vip-deployed.yaml \
  -e /home/stack/containers-prepare-parameter.yaml \
  -e /home/stack/inject-trust-anchor-hiera.yaml \
 [-r /home/stack/templates/roles_data.yaml ]
```

-n /home/stack/templates/network_data.yaml

Specifies the custom network configuration. Required if you use network isolation or custom composable networks. For information on configuring overcloud networks, see Configuring overcloud networking.

-e /home/stack/containers-prepare-parameter.yaml

Adds the container image preparation environment file. You generated this file during the undercloud installation and can use the same file for your overcloud creation.

-e /home/stack/inject-trust-anchor-hiera.yaml

Adds an environment file to install a custom certificate in the undercloud.

-r /home/stack/templates/roles_data.yaml

The generated roles data, if you use custom roles or want to enable a multi-architecture cloud.

4. When the overcloud creation completes, director provides a recap of the Ansible plays that were executed to configure the overcloud:

```
PLAY RECAP *************************************************************
overcloud-compute-0     : ok=160  changed=67   unreachable=0    failed=0
overcloud-controller-0  : ok=210  changed=93   unreachable=0    failed=0
undercloud              : ok=10   changed=7    unreachable=0    failed=0

Tuesday 15 October 2018  18:30:57 +1000 (0:00:00.107) 1:06:37.514 ******
========================================================================
```

5. When the overcloud creation completes, director provides details to access your overcloud:

```
Ansible passed.
```

```
Overcloud configuration completed.
Overcloud Endpoint: http://192.168.24.113:5000
Overcloud Horizon Dashboard URL: http://192.168.24.113:80/dashboard
Overcloud rc file: /home/stack/overcloudrc
Overcloud Deployed
```

TIP

You can keep your deployment command in a file that you add to every time you update your configuration with a new env file.

11.3.7. Deployment command options

The following table lists the additional parameters for the **openstack overcloud deploy** command.

IMPORTANT

Some options are available in this release as a *Technology Preview* and therefore are not fully supported by Red Hat. They should only be used for testing and should not be used in a production environment. For more information about Technology Preview features, see Scope of Coverage Details.

Table 11.8. Deployment command options

Parameter	Description
--templates [TEMPLATES]	The directory that contains the heat templates that you want to deploy. If blank, the deployment command uses the default template location at **/usr/share/openstack-tripleo-heat-templates/**
--stack STACK	The name of the stack that you want to create or update
-t [TIMEOUT], **--timeout [TIMEOUT]**	The deployment timeout duration in minutes
--libvirt-type [LIBVIRT_TYPE]	The virtualization type that you want to use for hypervisors
--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.

Parameter	Description
--no-proxy [NO_PROXY]	Defines custom values for the environment variable **no_proxy**, which excludes certain host names from proxy communication.
--overcloud-ssh-user OVERCLOUD_SSH_USER	Defines the SSH user to access the overcloud nodes. Normally SSH access occurs through the **tripleo-admin** user.
--overcloud-ssh-key OVERCLOUD_SSH_KEY	Defines the key path for SSH access to overcloud nodes.
--overcloud-ssh-network OVERCLOUD_SSH_NETWORK	Defines the network name that you want to use for SSH access to overcloud nodes.
-e [EXTRA HEAT TEMPLATE], **--environment-file [ENVIRONMENT FILE]**	Extra environment files that you want to pass to the overcloud deployment. You can specify this option more than once. Note that the order of environment files that you pass to the **openstack overcloud deploy** command is important. For example, parameters from each sequential environment file override the same parameters from earlier environment files.
--environment-directory	A directory that contains environment files that you want to include in deployment. The deployment command processes these environment files in numerical order, then alphabetical order.
-r ROLES_FILE	Defines the roles file and overrides the default **roles_data.yaml** in the **--templates** directory. The file location can be an absolute path or the path relative to **--templates**.
-n NETWORKS_FILE	Defines the networks file and overrides the default network_data.yaml in the **--templates** directory. The file location can be an absolute path or the path relative to **--templates**.
-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.

Parameter	Description
--update-plan-only	Use this option if you want to update the plan without performing the actual deployment.
--validation-errors-nonfatal	The overcloud creation process performs a set of pre-deployment checks. This option exits if any non-fatal errors occur from the pre-deployment checks. It is advisable to use this option as any errors can cause your deployment to fail.
--validation-warnings-fatal	The overcloud creation process performs a set of pre-deployment checks. This option exits if any non-critical warnings occur from the pre-deployment checks. openstack-tripleo-validations
--dry-run	Use this option if you want to perform a validation check on the overcloud without creating the overcloud.
--run-validations	Use this option to run external validations from the **openstack-tripleo-validations** package.
--skip-postconfig	Use this option to skip the overcloud post-deployment configuration.
--force-postconfig	Use this option to force the overcloud post-deployment configuration.
--skip-deploy-identifier	Use this option if you do not want the deployment command to generate a unique identifier for the **DeployIdentifier** parameter. The software configuration deployment steps only trigger if there is an actual change to the configuration. Use this option with caution and only if you are confident that you do not need to run the software configuration, such as scaling out certain roles.
--answers-file ANSWERS_FILE	The path to a YAML file with arguments and parameters.
--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**.

Parameter	Description
--no-config-download, --stack-only	Use this option if you want to disable the **config-download** workflow and create only the stack and associated OpenStack resources. This command applies no software configuration to the overcloud.
--config-download-only	Use this option if you want to disable the overcloud stack creation and only run the **config-download** workflow to apply the software configuration.
--output-dir OUTPUT_DIR	The directory that you want to use for saved **config-download** output. The directory must be writeable by the mistral user. When not specified, director uses the default, which is **/var/lib/mistral/overcloud**.
--override-ansible-cfg OVERRIDE_ANSIBLE_CFG	The path to an Ansible configuration file. The configuration in the file overrides any configuration that **config-download** generates by default.
--config-download-timeout CONFIG_DOWNLOAD_TIMEOUT	The timeout duration in minutes that you want to use for **config-download** steps. If unset, director sets the default to the amount of time remaining from the **--timeout** parameter after the stack deployment operation.
--limit NODE1,NODE2	**(Technology Preview)** Use this option with a comma-separated list of nodes to limit the config-download playbook execution to a specific node or set of nodes. For example, the **--limit** option can be useful for scale-up operations, when you want to run config-download only on new nodes. This argument might cause live migration of instances between hosts to fail, see Running config-download with the ansible-playbook-command.sh script
--tags TAG1,TAG2	**(Technology Preview)** Use this option with a comma-separated list of tags from the config-download playbook to run the deployment with a specific set of config-download tasks.
--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 11.9. 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**
--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

Parameter	Description	Heat template parameter
--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**
--reg-force	Use this option to register the system even if it is already registered.	**RhsmVars**

Parameter	Description	Heat template parameter
--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.

11.3.8. Validating your overcloud deployment

Validate your deployed overcloud.

Prerequisites

- You have deployed your overcloud.

Procedure

1. Source the **stackrc** credentials file:

```
$ source ~/stackrc
```

2. Validate your overcloud deployment:

```
$ validation run \
  --group post-deployment \
  [--inventory <inventory_file>]
```

- Replace **<inventory_file>** with the name of your ansible inventory file. By default, the dynamic inventory is called **tripleo-ansible-inventory**.

  NOTE

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

3. Review the results of the validation report:

```
$ validation show run [--full] <UUID>
```

- Replace **<UUID>** with the UUID of the validation run.
- Optional: Use the **--full** option to view detailed output from the validation run.

IMPORTANT

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

Addtional resources

- Using the validation framework

11.3.9. Accessing the overcloud

Director generates a credential file containing the credentials necessary to operate the overcloud from the undercloud. Director saves this file, **overcloudrc**, in the home directory of the **stack** user.

Procedure

1. Source the **overcloudrc** file:

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

   The command prompt changes to indicate that you are accessing the overcloud:

```
(overcloud)$
```

2. To return to interacting with the undercloud, source the **stackrc** file:

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

   The command prompt changes to indicate that you are accessing the undercloud:

```
(undercloud)$
```

11.4. CONFIGURING A BASIC OVERCLOUD WITH PRE-PROVISIONED NODES

This chapter contains basic configuration procedures that you can use to configure a Red Hat OpenStack Platform (RHOSP) environment with pre-provisioned nodes. This scenario differs from the standard overcloud creation scenarios in several ways:

- You can provision nodes with an external tool and let the director control the overcloud configuration only.
- You can use nodes without relying on the director provisioning methods. This is useful if you want to create an overcloud without power management control, or use networks with DHCP/PXE boot restrictions.
- The director does not use OpenStack Compute (nova), OpenStack Bare Metal (ironic), or OpenStack Image (glance) to manage nodes.
- Pre-provisioned nodes can use a custom partitioning layout that does not rely on the QCOW2 overcloud-full image.

This scenario includes only basic configuration with no custom features.

IMPORTANT

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

11.4.1. Pre-provisioned node requirements

Before you begin deploying an overcloud with pre-provisioned nodes, ensure that the following configuration is present in your environment:

- The director node that you created in Chapter 7, *Installing director on the undercloud* .
- A set of bare metal machines for your nodes. The number of nodes required depends on the type of overcloud you intend to create. These machines must comply with the requirements set for each node type. These nodes require Red Hat Enterprise Linux 9.0 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 11.10. Provisioning Network IP assignments

Node name	IP address
Director	192.168.24.1
Controller 0	192.168.24.2

Node name	IP address
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 11.4.6, "Using a separate network for pre-provisioned nodes".

- All other network types in this example also use the Control Plane network for OpenStack services. However, you can create additional networks for other network traffic types.

- If any nodes use Pacemaker resources, the service user **hacluster** and the service group **haclient** must have a UID/GID of 189. This is due to CVE-2018-16877. If you installed Pacemaker together with the operating system, the installation creates these IDs automatically. If the ID values are set incorrectly, follow the steps in the article OpenStack minor update / fast-forward upgrade can fail on the controller nodes at pacemaker step with "Could not evaluate: backup_cib" to change the ID values.

- To prevent some services from binding to an incorrect IP address and causing deployment failures, make sure that the **/etc/hosts** file does not include the **node-name=127.0.0.1** mapping.

11.4.2. Creating a user on pre-provisioned nodes

When you configure an overcloud with pre-provisioned nodes, director requires SSH access to the overcloud nodes. On the pre-provisioned nodes, you must create a user with SSH key authentication and configure passwordless sudo access for that user. After you create a user on pre-provisioned nodes, you can use the **--overcloud-ssh-user** and **--overcloud-ssh-key** options with the **openstack overcloud deploy** command to create an overcloud with pre-provisioned nodes.

By default, the values for the overcloud SSH user and overcloud SSH key are the **stack** user and **~/.ssh/id_rsa**. To create the **stack** user, complete the following steps.

Procedure

1. On each overcloud node, create the **stack** user and set a password. For example, run the following commands on the Controller node:

```
[root@controller-0 ~]# useradd stack
[root@controller-0 ~]# passwd stack  # specify a password
```

2. Disable password requirements for this user when using **sudo**:

```
[root@controller-0 ~]# echo "stack ALL=(root) NOPASSWD:ALL" | tee -a
/etc/sudoers.d/stack
[root@controller-0 ~]# chmod 0440 /etc/sudoers.d/stack
```

3. After you create and configure the **stack** user on all pre-provisioned nodes, copy the **stack** user's public SSH key from the director node to each overcloud node. For example, to copy the director's public SSH key to the Controller node, run the following command:

```
[stack@director ~]$ ssh-copy-id stack@192.168.24.2
```

IMPORTANT

To copy your SSH keys, you might have to temporarily set **PasswordAuthentication Yes** in the SSH configuration of each overcloud node. After you copy the SSH keys, set **PasswordAuthentication No** and use the SSH keys to authenticate in the future.

11.4.3. Registering the operating system for pre-provisioned nodes

Each node requires access to a Red Hat subscription. Complete the following steps on each node to register your nodes with the Red Hat Content Delivery Network. Execute the commands as the **root** user or as a user with **sudo** privileges.

IMPORTANT

Enable only the repositories listed. Additional repositories can cause package and software conflicts. Do not enable any additional repositories.

Procedure

1. Run the registration command and enter your Customer Portal user name and password when prompted:

```
[root@controller-0 ~]# sudo subscription-manager register
```

2. Find the entitlement pool for Red Hat OpenStack Platform 17.0:

```
[root@controller-0 ~]# sudo subscription-manager list --available --all --matches="Red Hat
OpenStack"
```

3. Use the pool ID located in the previous step to attach the Red Hat OpenStack Platform 16 entitlements:

```
[root@controller-0 ~]# sudo subscription-manager attach --pool=pool_id
```

4. Disable all default repositories:

```
[root@controller-0 ~]# sudo subscription-manager repos --disable=*
```

5. Enable the required Red Hat Enterprise Linux repositories:

```
[root@controller-0 ~]# sudo subscription-manager repos \
 --enable=rhel-9-for-x86_64-baseos-eus-rpms \
 --enable=rhel-9-for-x86_64-appstream-eus-rpms \
 --enable=rhel-9-for-x86_64-highavailability-eus-rpms \
 --enable=openstack-beta-for-rhel-9-x86_64-rpms \
 --enable=fast-datapath-for-rhel-9-x86_64-rpms
```

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

```
[root@cephstorage-0 ~]# sudo subscription-manager repos \
--enable=rhel-9-for-x86_64-baseos-rpms \
--enable=rhel-9-for-x86_64-appstream-rpms \
--enable=openstack-beta-deployment-tools-for-rhel-9-x86_64-rpms
```

7. Lock the RHEL version on all overcloud nodes except Red Hat Ceph Storage nodes:

   ```
   [root@controller-0 ~]# sudo subscription-manager release --set=9.0
   ```

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

   ```
   [root@controller-0 ~]# sudo dnf update -y
   [root@controller-0 ~]# sudo reboot
   ```

The node is now ready to use for your overcloud.

11.4.4. Configuring SSL/TLS access to director

If the director uses SSL/TLS, the pre-provisioned nodes require the certificate authority file used to sign the director's SSL/TLS certificates. If you use your own certificate authority, perform the following actions on each overcloud node.

Procedure

1. Copy the certificate authority file to the **/etc/pki/ca-trust/source/anchors/** directory on each pre-provisioned node.

2. Run the following command on each overcloud node:

   ```
   [root@controller-0 ~]#  sudo update-ca-trust extract
   ```

These steps ensure that the overcloud nodes can access the director's Public API over SSL/TLS.

11.4.5. Configuring networking for the control plane

The pre-provisioned overcloud nodes obtain metadata from director using standard HTTP requests. This means all overcloud nodes require L3 access to either:

- The director Control Plane network, which is the subnet that you define with the **network_cidr** parameter in your **undercloud.conf** file. The overcloud nodes require either direct access to this subnet or routable access to the subnet.

- The director Public API endpoint, that you specify with the **undercloud_public_host** parameter in your **undercloud.conf** file. This option is available if you do not have an L3 route to the Control Plane or if you want to use SSL/TLS communication. For more information about configuring your overcloud nodes to use the Public API endpoint, see Section 11.4.6, "Using a separate network for pre-provisioned nodes".

Director uses the Control Plane network to manage and configure a standard overcloud. For an overcloud with pre-provisioned nodes, your network configuration might require some modification to accommodate communication between the director and the pre-provisioned nodes.

Using network isolation

You can use network isolation to group services to use specific networks, including the Control Plane. You can also define specific IP addresses for nodes on the Control Plane. For more information about isolating networks and creating predictable node placement strategies, see Network isolation.

NOTE

If you use network isolation, ensure that your NIC templates do not include the NIC used for undercloud access. These templates can reconfigure the NIC, which introduces connectivity and configuration problems during deployment.

Assigning IP addresses

If you do not use network isolation, you can use a single Control Plane network to manage all services. This requires manual configuration of the Control Plane NIC on each node to use an IP address within the Control Plane network range. If you are using the director Provisioning network as the Control Plane, ensure that the overcloud IP addresses that you choose are outside of the DHCP ranges for both provisioning (**dhcp_start** and **dhcp_end**) and introspection (**inspection_iprange**).

During standard overcloud creation, director creates OpenStack Networking (neutron) ports and automatically assigns IP addresses to the overcloud nodes on the Provisioning / Control Plane network. However, this can cause director to assign different IP addresses to the ones that you configure manually for each node. In this situation, use a predictable IP address strategy to force director to use the pre-provisioned IP assignments on the Control Plane.

If you are using network isolation, create a custom environment file, **deployed-ports.yaml**, to implement a predictable IP strategy. The following example custom environment file, **deployed-ports.yaml**, passes a set of resource registry mappings and parameters to director, and defines the IP assignments of the pre-provisioned nodes. The **NodePortMap**, **ControlPlaneVipData**, and **VipPortMap** parameters define the IP addresses and subnet CIDRs that correspond to each overcloud node.

```
resource_registry:
  # Deployed Virtual IP port resources
  OS::TripleO::Network::Ports::ExternalVipPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/deployed_vip_external.yaml
  OS::TripleO::Network::Ports::InternalApiVipPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/deployed_vip_internal_api.yaml
  OS::TripleO::Network::Ports::StorageVipPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/deployed_vip_storage.yaml
  OS::TripleO::Network::Ports::StorageMgmtVipPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/deployed_vip_storage_mgmt.yaml

  # Deployed ControlPlane port resource
  OS::TripleO::DeployedServer::ControlPlanePort: /usr/share/openstack-tripleo-heat-
templates/deployed-server/deployed-neutron-port.yaml

  # Controller role port resources
  OS::TripleO::Controller::Ports::ExternalPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/deployed_external.yaml
  OS::TripleO::Controller::Ports::InternalApiPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/deployed_internal_api.yaml
  OS::TripleO::Controller::Ports::StorageMgmtPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/deployed_storage_mgmt.yaml
  OS::TripleO::Controller::Ports::StoragePort: /usr/share/openstack-tripleo-heat-
templates/network/ports/deployed_storage.yaml
  OS::TripleO::Controller::Ports::TenantPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/deployed_tenant.yaml
```

```
  # Compute role port resources
  OS::TripleO::Compute::Ports::InternalApiPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/deployed_internal_api.yaml
  OS::TripleO::Compute::Ports::StoragePort: /usr/share/openstack-tripleo-heat-
templates/network/ports/deployed_storage.yaml
  OS::TripleO::Compute::Ports::TenantPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/deployed_tenant.yaml

  # CephStorage role port resources
  OS::TripleO::CephStorage::Ports::StorageMgmtPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/deployed_storage_mgmt.yaml
  OS::TripleO::CephStorage::Ports::StoragePort: /usr/share/openstack-tripleo-heat-
templates/network/ports/deployed_storage.yaml

parameter_defaults:
  NodePortMap: 1
    # Controller node parameters
    controller-00-rack01: 2
      ctlplane: 3
        ip_address: 192.168.24.201
        ip_address_uri: 192.168.24.201
        ip_subnet: 192.168.24.0/24
      external:
        ip_address: 10.0.0.201
        ip_address_uri: 10.0.0.201
        ip_subnet: 10.0.0.10/24
      internal_api:
        ip_address: 172.16.2.201
        ip_address_uri: 172.16.2.201
        ip_subnet: 172.16.2.10/24
      management:
        ip_address: 192.168.1.201
        ip_address_uri: 192.168.1.201
        ip_subnet: 192.168.1.10/24
      storage:
        ip_address: 172.16.1.201
        ip_address_uri: 172.16.1.201
        ip_subnet: 172.16.1.10/24
      storage_mgmt:
        ip_address: 172.16.3.201
        ip_address_uri: 172.16.3.201
        ip_subnet: 172.16.3.10/24
      tenant:
        ip_address: 172.16.0.201
        ip_address_uri: 172.16.0.201
        ip_subnet: 172.16.0.10/24
    ...

    # Compute node parameters
    compute-00-rack01:
      ctlplane:
        ip_address: 192.168.24.11
        ip_address_uri: 192.168.24.11
        ip_subnet: 192.168.24.0/24
      internal_api:
        ip_address: 172.16.2.11
        ip_address_uri: 172.16.2.11
        ip_subnet: 172.16.2.10/24
      storage:
        ip_address: 172.16.1.11
        ip_address_uri: 172.16.1.11
        ip_subnet: 172.16.1.10/24
      tenant:
        ip_address: 172.16.0.11
        ip_address_uri: 172.16.0.11
        ip_subnet: 172.16.0.10/24
    ...

    # Ceph node parameters
    ceph-00-rack01:
      ctlplane:
        ip_address: 192.168.24.101
        ip_address_uri: 192.168.24.101
        ip_subnet: 192.168.24.0/24
      storage:
        ip_address: 172.16.1.101
        ip_address_uri: 172.16.1.101
        ip_subnet: 172.16.1.10/24
      storage_mgmt:
        ip_address: 172.16.3.101
        ip_address_uri: 172.16.3.101
        ip_subnet: 172.16.3.10/24
    ...

  # Virtual IP address parameters
  ControlPlaneVipData:
    fixed_ips:
    - ip_address: 192.168.24.5
    name: control_virtual_ip
    network:
      tags: [192.168.24.0/24]
      subnets:
      - ip_version: 4
  VipPortMap
    external:
      ip_address: 10.0.0.100
      ip_address_uri: 10.0.0.100
      ip_subnet: 10.0.0.100/24
    internal_api:
      ip_address: 172.16.2.100
      ip_address_uri: 172.16.2.100
      ip_subnet: 172.16.2.100/24
    storage:
      ip_address: 172.16.1.100
      ip_address_uri: 172.16.1.100
      ip_subnet: 172.16.1.100/24
    storage_mgmt:
      ip_address: 172.16.3.100
      ip_address_uri: 172.16.3.100
      ip_subnet: 172.16.3.100/24
```

```
RedisVirtualFixedIPs:
  - ip_address: 192.168.24.6
    use_neutron: false
```

1. The **NodePortMap** mappings define the names of the node assignments.
2. The short host name for the node, which follows the format **<node_hostname>**.
3. The network definitions and IP assignments for the node. Networks include **ctlplane**, **external**, **internal_api**, **management**, **storage**, **storage_mgmt**, and **tenant**. The IP assignments include the **ip_address**, the **ip_address_uri**, and the **ip_subnet**:
   - IPv4: **ip_address** and **ip_address_uri** should be set to the same value.
   - IPv6:
     - **ip_address**: Set to the IPv6 address without brackets.
     - **ip_address_uri**: Set to the IPv6 address in square brackets, for example, **[2001:0db8:85a3:0000:0000:8a2e:0370:7334]**.

11.4.6. Using a separate network for pre-provisioned nodes

By default, director uses the Provisioning network as the overcloud Control Plane. However, if this network is isolated and non-routable, nodes cannot communicate with the director Internal API during configuration. In this situation, you might need to define a separate network for the nodes and configure them to communicate with the director over the Public API.

There are several requirements for this scenario:

- The overcloud nodes must accommodate the basic network configuration from Section 11.4.5, "Configuring networking for the control plane".
- You must enable SSL/TLS on the director for Public API endpoint usage. For more information, see Enabling SSL/TLS on overcloud public endpoints.
- You must define an accessible fully qualified domain name (FQDN) for director. This FQDN must resolve to a routable IP address for the director. Use the **undercloud_public_host** parameter in the **undercloud.conf** file to set this FQDN.

The examples in this section use IP address assignments that differ from the main scenario:

Table 11.11. 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

Node Name	IP address or FQDN
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 11.4.5, “Configuring networking for the control plane”. However, since the Control Plane may not be routable from the deployed servers, you can use the **NodePortMap**, **ControlPlaneVipData**, and **VipPortMap** parameters to assign IP addresses from your chosen overcloud node subnet, including the virtual IP address to access the Control Plane. The following example is a modified version of the **deployed-ports.yaml** custom environment file from Section 11.4.5, “Configuring networking for the control plane” that accommodates this network architecture:

```
parameter_defaults:
  NodePortMap:
    controller-00-rack01
      ctlplane
        ip_address: 192.168.100.2
        ip_address_uri: 192.168.100.2
        ip_subnet: 192.168.100.0/24
...
    compute-00-rack01:
      ctlplane
        ip_address: 192.168.100.3
        ip_address_uri: 192.168.100.3
        ip_subnet: 192.168.100.0/24
...
  ControlPlaneVipData:
    fixed_ips:
    - ip_address: 192.168.100.1
    name: control_virtual_ip
    network:
      tags: [192.168.100.0/24]
    subnets:
    - ip_version: 4
  VipPortMap:
    external:
      ip_address: 10.0.0.100
      ip_address_uri: 10.0.0.100
      ip_subnet: 10.0.0.100/24
....
  RedisVirtualFixedIPs: 1
    - ip_address: 192.168.100.10
      use_neutron: false
```

1 The **RedisVipPort** resource is mapped to **network/ports/noop.yaml**. This mapping is necessary because the default Redis VIP address comes from the Control Plane. In this situation, use a **noop** to disable this Control Plane mapping.

11.4.7. Mapping pre-provisioned node hostnames

When you configure pre-provisioned nodes, you must map heat-based hostnames to their actual hostnames so that **ansible-playbook** can reach a resolvable host. Use the **HostnameMap** to map these values.

Procedure

1. Create an environment file, for example **hostname-map.yaml**, and include the **HostnameMap** parameter and the hostname mappings. Use the following syntax:

   ```
   parameter_defaults:
     HostnameMap:
       [HEAT HOSTNAME]: [ACTUAL HOSTNAME]
       [HEAT HOSTNAME]: [ACTUAL HOSTNAME]
   ```

   The **[HEAT HOSTNAME]** usually conforms to the following convention: **[STACK NAME]-[ROLE]-[INDEX]**:

   ```
   parameter_defaults:
     HostnameMap:
       overcloud-controller-0: controller-00-rack01
       overcloud-controller-1: controller-01-rack02
       overcloud-controller-2: controller-02-rack03
       overcloud-novacompute-0: compute-00-rack01
       overcloud-novacompute-1: compute-01-rack01
       overcloud-novacompute-2: compute-02-rack01
   ```

2. Save the **hostname-map.yaml** file.

11.4.8. Configuring Ceph Storage for pre-provisioned nodes

Complete the following steps on the undercloud host to configure Ceph for nodes that are already deployed.

Procedure

1. On the undercloud host, create an environment variable, **OVERCLOUD_HOSTS**, and set the variable to a space-separated list of IP addresses of the overcloud hosts that you want to use as Ceph clients:

   ```
   export OVERCLOUD_HOSTS="192.168.1.8 192.168.1.42"
   ```

2. The default overcloud plan name is **overcloud**. If you use a different name, create an environment variable **OVERCLOUD_PLAN** to store your custom name:

   ```
   export OVERCLOUD_PLAN="<custom-stack-name>"
   ```

   - Replace **<custom-stack-name>** with the name of your stack.

3. Run the **enable-ssh-admin.sh** script to configure a user on the overcloud nodes that Ansible can use to configure Ceph clients:

```
bash /usr/share/openstack-tripleo-heat-templates/deployed-server/scripts/enable-ssh-
admin.sh
```

When you run the **openstack overcloud deploy** command, Ansible configures the hosts that you define in the **OVERCLOUD_HOSTS** variable as Ceph clients.

11.4.9. Creating the overcloud with pre-provisioned nodes

The overcloud deployment uses the standard CLI methods. For pre-provisioned nodes, the deployment command requires some additional options and environment files from the core heat template collection:

- **--disable-validations** - Use this option to disable basic CLI validations for services not used with pre-provisioned infrastructure. If you do not disable these validations, the deployment fails.
- **environments/deployed-server-environment.yaml** - Include this environment file to create and configure the pre-provisioned infrastructure. This environment file substitutes the **OS::Nova::Server** resources with **OS::Heat::DeployedServer** resources.

The following command is an example overcloud deployment command with the environment files specific to the pre-provisioned architecture:

```
$ source ~/stackrc
(undercloud)$ openstack overcloud deploy \
  --disable-validations \
  -e /usr/share/openstack-tripleo-heat-templates/environments/deployed-server-environment.yaml \
  -e /home/stack/templates/deployed-ports.yaml \
  -e /home/stack/templates/hostname-map.yaml \
  --overcloud-ssh-user stack \
  --overcloud-ssh-key ~/.ssh/id_rsa \
  <OTHER OPTIONS>
```

The **--overcloud-ssh-user** and **--overcloud-ssh-key** options are used to SSH into each overcloud node during the configuration stage, create an initial **tripleo-admin** user, and inject an SSH key into **/home/tripleo-admin/.ssh/authorized_keys**. To inject the SSH key, specify the credentials for the initial SSH connection with **--overcloud-ssh-user** and **--overcloud-ssh-key** (defaults to **~/.ssh/id_rsa**). To limit exposure to the private key that you specify with the **--overcloud-ssh-key** option, director never passes this key to any API service, such as heat, and only the director **openstack overcloud deploy** command uses this key to enable access for the **tripleo-admin** user.

11.4.10. Accessing the overcloud

Director generates a credential file containing the credentials necessary to operate the overcloud from the undercloud. Director saves this file, **overcloudrc**, in the home directory of the **stack** user.

Procedure

1. Source the **overcloudrc** file:

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

   The command prompt changes to indicate that you are accessing the overcloud:

   ```
   (overcloud)$
   ```

2. To return to interacting with the undercloud, source the **stackrc** file:

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

   The command prompt changes to indicate that you are accessing the undercloud:

   ```
   (undercloud)$
   ```

11.4.11. Scaling pre-provisioned nodes

The process for scaling pre-provisioned nodes is similar to the standard scaling procedures in Chapter 19, *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 11.4.1, "Pre-provisioned node requirements".

2. Scale up the nodes. For more information, see Chapter 19, *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 19, *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 15.7, “Removing an overcloud stack” for the standard overcloud removal procedure. After you remove the overcloud, power off all nodes and reprovision them to a base operating system configuration.

NOTE

Do not attempt to reuse nodes previously removed from the overcloud without first reprovisioning them with a fresh base operating system. The removal process only deletes the overcloud stack and does not uninstall any packages.

CHAPTER 12. ANSIBLE-BASED OVERCLOUD REGISTRATION

Director uses Ansible-based methods to register overcloud nodes to the Red Hat Customer Portal or to a Red Hat Satellite Server.

I:f you used the **rhel-registration** method from previous Red Hat OpenStack Platform versions, you must disable it and switch to the Ansible-based method. For more information, see Section 12.6, "Switching to the rhsm composable service" and Section 12.7, "rhel-registration to rhsm mappings" .

In addition to the director-based registration method, you can also manually register after deployment. For more information, see Section 12.9, "Running Ansible-based registration manually"

12.1. RED HAT SUBSCRIPTION MANAGER (RHSM) COMPOSABLE SERVICE

You can use the **rhsm** composable service to register overcloud nodes through Ansible. Each role in the default **roles_data** file contains a **OS::TripleO::Services::Rhsm** resource, which is disabled by default. To enable the service, register the resource to the **rhsm** composable service file:

```
resource_registry:
  OS::TripleO::Services::Rhsm: /usr/share/openstack-tripleo-heat-templates/deployment/rhsm/rhsm-
baremetal-ansible.yaml
```

The **rhsm** composable service accepts a **RhsmVars** parameter, which you can use to define multiple sub-parameters relevant to your registration:

```
parameter_defaults:
  RhsmVars:
    rhsm_repos:
      - rhel-9-for-x86_64-baseos-eus-rpms
      - rhel-9-for-x86_64-appstream-eus-rpms
      - rhel-9-for-x86_64-highavailability-eus-rpms
      …
    rhsm_username: "myusername"
    rhsm_password: "p@55w0rd!"
    rhsm_org_id: "1234567"
    rhsm_release: 9.0
```

You can also use the **RhsmVars** parameter in combination with role-specific parameters, for example, **ControllerParameters**, to provide flexibility when enabling specific repositories for different nodes types.

12.2. RHSMVARS SUB-PARAMETERS

Use the following sub-parameters as part of the **RhsmVars** parameter when you configure the **rhsm** composable service. For more information about the Ansible parameters that are available, see the role documentation.

rhsm	Description
rhsm_method	Choose the registration method. Either **portal**, **satellite**, or **disable**.

rhsm	Description
rhsm_org_id	The organization that you want to use for registration. To locate this ID, run **sudo subscription-manager orgs** from the undercloud node. Enter your Red Hat credentials at the prompt, and use the resulting **Key** value. For more information on your organization ID, see Understanding the Red Hat Subscription Management Organization ID.
rhsm_pool_ids	The subscription pool ID that you want to use. Use this parameter if you do not want to auto-attach subscriptions. To locate this ID, run **sudo subscription-manager list --available --all --matches="*Red Hat OpenStack*"** from the undercloud node, and use the resulting**Pool ID** value.
rhsm_activation_key	The activation key that you want to use for registration.
rhsm_autosubscribe	Use this parameter to attach compatible subscriptions to this system automatically. Set the value to **true** to enable this feature.
rhsm_baseurl	The base URL for obtaining content. The default URL is the Red Hat Content Delivery Network. If you use a Satellite server, change this value to the base URL of your Satellite server content repositories.
rhsm_server_hostname	The hostname of the subscription management service for registration. The default is the Red Hat Subscription Management hostname. If you use a Satellite server, change this value to your Satellite server hostname.
rhsm_repos	A list of repositories that you want to enable.
rhsm_username	The username for registration. If possible, use activation keys for registration.
rhsm_password	The password for registration. If possible, use activation keys for registration.
rhsm_release	Red Hat Enterprise Linux release for pinning the repositories. This is set to 9.0 for Red Hat OpenStack Platform
rhsm_rhsm_proxy_hostname	The hostname for the HTTP proxy. For example: **proxy.example.com**.
rhsm_rhsm_proxy_port	The port for HTTP proxy communication. For example: **8080**.
rhsm_rhsm_proxy_user	The username to access the HTTP proxy.
rhsm_rhsm_proxy_password	The password to access the HTTP proxy.

IMPORTANT

You can use **rhsm_activation_key** and **rhsm_repos** together only if **rhsm_method** is set to **portal**. If **rhsm_method** is set to *satellite*, you can only use either **rhsm_activation_key** or **rhsm_repos**.

12.3. REGISTERING THE OVERCLOUD WITH THE RHSM COMPOSABLE SERVICE

Create an environment file that enables and configures the **rhsm** composable service. Director uses this environment file to register and subscribe your nodes.

Procedure

1. Create an environment file named **templates/rhsm.yml** to store the configuration.

2. Include your configuration in the environment file. For example:

```
resource_registry:
  OS::TripleO::Services::Rhsm: /usr/share/openstack-tripleo-heat-
templates/deployment/rhsm/rhsm-baremetal-ansible.yaml
parameter_defaults:
  RhsmVars:
    rhsm_repos:
      - rhel-9-for-x86_64-baseos-eus-rpms
      - rhel-9-for-x86_64-appstream-eus-rpms
      - rhel-9-for-x86_64-highavailability-eus-rpms
      …
    rhsm_username: "myusername"
    rhsm_password: "p@55w0rd!"
    rhsm_org_id: "1234567"
    rhsm_pool_ids: "1a85f9223e3d5e43013e3d6e8ff506fd"
    rhsm_method: "portal"
    rhsm_release: 9.0
```

- The **resource_registry** section associates the **rhsm** composable service with the **OS::TripleO::Services::Rhsm** resource, which is available on each role.

- The **RhsmVars** variable passes parameters to Ansible for configuring your Red Hat registration.

3. Save the environment file.

12.4. APPLYING THE RHSM COMPOSABLE SERVICE TO DIFFERENT ROLES

You can apply the **rhsm** composable service on a per-role basis. For example, you can apply different sets of configurations to Controller nodes, Compute nodes, and Ceph Storage nodes.

Procedure

1. Create an environment file named **templates/rhsm.yml** to store the configuration.

2. Include your configuration in the environment file. For example:

```
resource_registry:
  OS::TripleO::Services::Rhsm: /usr/share/openstack-tripleo-heat-
templates/deployment/rhsm/rhsm-baremetal-ansible.yaml
parameter_defaults:
  ControllerParameters:
    RhsmVars:
      rhsm_repos:
        - rhel-9-for-x86_64-baseos-eus-rpms
        - rhel-9-for-x86_64-appstream-eus-rpms
        - rhel-9-for-x86_64-highavailability-eus-rpms
        - openstack-17-for-rhel-9-x86_64-rpms
        - fast-datapath-for-rhel-9-x86_64-rpms
        - rhceph-5-tools-for-rhel-9-x86_64-rpms
      rhsm_username: "myusername"
      rhsm_password: "p@55w0rd!"
      rhsm_org_id: "1234567"
      rhsm_pool_ids: "55d251f1490556f3e75aa37e89e10ce5"
      rhsm_method: "portal"
      rhsm_release: 9.0
  ComputeParameters:
    RhsmVars:
      rhsm_repos:
        - rhel-9-for-x86_64-baseos-eus-rpms
        - rhel-9-for-x86_64-appstream-eus-rpms
        - rhel-9-for-x86_64-highavailability-eus-rpms
        - openstack-17-for-rhel-9-x86_64-rpms
        - rhceph-5-tools-for-rhel-9-x86_64-rpms
        - fast-datapath-for-rhel-9-x86_64-rpms
      rhsm_username: "myusername"
      rhsm_password: "p@55w0rd!"
      rhsm_org_id: "1234567"
      rhsm_pool_ids: "55d251f1490556f3e75aa37e89e10ce5"
      rhsm_method: "portal"
      rhsm_release: 9.0
  CephStorageParameters:
    RhsmVars:
      rhsm_repos:
        - rhel-9-for-x86_64-baseos-rpms
        - rhel-9-for-x86_64-appstream-rpms
        - rhel-9-for-x86_64-highavailability-rpms
        - openstack-17-deployment-tools-for-rhel-9-x86_64-rpms
      rhsm_username: "myusername"
      rhsm_password: "p@55w0rd!"
      rhsm_org_id: "1234567"
      rhsm_pool_ids: "68790a7aa2dc9dc50a9bc39fabc55e0d"
      rhsm_method: "portal"
      rhsm_release: 9.0
```

The **resource_registry** associates the **rhsm** composable service with the **OS::TripleO::Services::Rhsm** resource, which is available on each role.

The **ControllerParameters**, **ComputeParameters**, and **CephStorageParameters** parameters each use a separate **RhsmVars** parameter to pass subscription details to their respective roles.

NOTE

Set the **RhsmVars** parameter within the **CephStorageParameters** parameter to use a Red Hat Ceph Storage subscription and repositories specific to Ceph Storage. Ensure the **rhsm_repos** parameter contains the standard Red Hat Enterprise Linux repositories instead of the Extended Update Support (EUS) repositories that Controller and Compute nodes require.

3. Save the environment file.

12.5. REGISTERING THE OVERCLOUD TO RED HAT SATELLITE SERVER

Create an environment file that enables and configures the **rhsm** composable service to register nodes to Red Hat Satellite instead of the Red Hat Customer Portal.

Procedure

1. Create an environment file named **templates/rhsm.yml** to store the configuration.

2. Include your configuration in the environment file. For example:

```
resource_registry:
  OS::TripleO::Services::Rhsm: /usr/share/openstack-tripleo-heat-
templates/deployment/rhsm/rhsm-baremetal-ansible.yaml
parameter_defaults:
  RhsmVars:
    rhsm_activation_key: "myactivationkey"
    rhsm_method: "satellite"
    rhsm_org_id: "ACME"
    rhsm_server_hostname: "satellite.example.com"
    rhsm_baseurl: "https://satellite.example.com/pulp/repos"
    rhsm_release: 9.0
```

The **resource_registry** associates the **rhsm** composable service with the **OS::TripleO::Services::Rhsm** resource, which is available on each role.

The **RhsmVars** variable passes parameters to Ansible for configuring your Red Hat registration.

3. Save the environment file.

12.6. SWITCHING TO THE RHSM COMPOSABLE SERVICE

The previous **rhel-registration** method runs a bash script to handle the overcloud registration. The scripts and environment files for this method are located in the core heat template collection at **/usr/share/openstack-tripleo-heat-templates/extraconfig/pre_deploy/rhel-registration/**.

Complete the following steps to switch from the **rhel-registration** method to the **rhsm** composable service.

Procedure

1. Exclude the **rhel-registration** environment files from future deployments operations. In most cases, exclude the following files:

- **rhel-registration/environment-rhel-registration.yaml**
- **rhel-registration/rhel-registration-resource-registry.yaml**

2. If you use a custom **roles_data** file, ensure that each role in your **roles_data** file contains the **OS::TripleO::Services::Rhsm** composable service. For example:

```
- name: Controller
  description: |
    Controller role that has all the controller services loaded and handles
    Database, Messaging and Network functions.
  CountDefault: 1
  ...
  ServicesDefault:
    ...
    - OS::TripleO::Services::Rhsm
    ...
```

3. Add the environment file for **rhsm** composable service parameters to future deployment operations.

This method replaces the **rhel-registration** parameters with the **rhsm** service parameters and changes the heat resource that enables the service from:

```
resource_registry:
  OS::TripleO::NodeExtraConfig: rhel-registration.yaml
```

To:

```
resource_registry:
  OS::TripleO::Services::Rhsm: /usr/share/openstack-tripleo-heat-templates/deployment/rhsm/rhsm-baremetal-ansible.yaml
```

You can also include the **/usr/share/openstack-tripleo-heat-templates/environments/rhsm.yaml** environment file with your deployment to enable the service.

12.7. RHEL-REGISTRATION TO RHSM MAPPINGS

To help transition your details from the **rhel-registration** method to the **rhsm** method, use the following table to map your parameters and values.

rhel-registration	rhsm / RhsmVars
rhel_reg_method	**rhsm_method**
rhel_reg_org	**rhsm_org_id**
rhel_reg_pool_id	**rhsm_pool_ids**
rhel_reg_activation_key	**rhsm_activation_key**
rhel_reg_auto_attach	**rhsm_autosubscribe**

rhel-registration	rhsm / RhsmVars
rhel_reg_sat_url	**rhsm_satellite_url**
rhel_reg_repos	**rhsm_repos**
rhel_reg_user	**rhsm_username**
rhel_reg_password	**rhsm_password**
rhel_reg_release	**rhsm_release**
rhel_reg_http_proxy_host	**rhsm_rhsm_proxy_hostname**
rhel_reg_http_proxy_port	**rhsm_rhsm_proxy_port**
rhel_reg_http_proxy_username	**rhsm_rhsm_proxy_user**
rhel_reg_http_proxy_password	**rhsm_rhsm_proxy_password**

12.8. DEPLOYING THE OVERCLOUD WITH THE RHSM COMPOSABLE SERVICE

Deploy the overcloud with the **rhsm** composable service so that Ansible controls the registration process for your overcloud nodes.

Procedure

1. Include **rhsm.yml** environment file with the **openstack overcloud deploy** command:

```
openstack overcloud deploy \
    <other cli args> \
    -e ~/templates/rhsm.yaml
```

   This enables the Ansible configuration of the overcloud and the Ansible-based registration.

2. Wait until the overcloud deployment completes.

3. Check the subscription details on your overcloud nodes. For example, log in to a Controller node and run the following commands:

```
$ sudo subscription-manager status
$ sudo subscription-manager list --consumed
```

12.9. RUNNING ANSIBLE-BASED REGISTRATION MANUALLY

You can perform manual Ansible-based registration on a deployed overcloud with the dynamic inventory script on the director node. Use this script to define node roles as host groups and then run a playbook against them with **ansible-playbook**. Use the following example playbook to register Controller nodes manually.

Procedure

1. Create a playbook that uses the **redhat_subscription** modules to register your nodes. For example, the following playbook applies to Controller nodes:

```
---
- name: Register Controller nodes
  hosts: Controller
  become: yes
  vars:
    repos:
      - rhel-9-for-x86_64-baseos-eus-rpms
      - rhel-9-for-x86_64-appstream-eus-rpms
      - rhel-9-for-x86_64-highavailability-eus-rpms
      - openstack-17-for-rhel-9-x86_64-rpms
      - fast-datapath-for-rhel-9-x86_64-rpms
  tasks:
    - name: Register system
      redhat_subscription:
        username: myusername
        password: p@55w0rd!
        org_id: 1234567
        release: 9.0
        pool_ids: 1a85f9223e3d5e43013e3d6e8ff506fd
    - name: Disable all repos
      command: "subscription-manager repos --disable *"
    - name: Enable Controller node repos
      command: "subscription-manager repos --enable {{ item }}"
      with_items: "{{ repos }}"
```

   - This play contains three tasks:
     - Register the node.
     - Disable any auto-enabled repositories.
     - Enable only the repositories relevant to the Controller node. The repositories are listed with the **repos** variable.

2. After you deploy the overcloud, you can run the following command so that Ansible executes the playbook (**ansible-osp-registration.yml**) against your overcloud:

```
$ ansible-playbook -i /usr/bin/tripleo-ansible-inventory ansible-osp-registration.yml
```

   This command performs the following actions:

   - Runs the dynamic inventory script to get a list of host and their groups.
   - Applies the playbook tasks to the nodes in the group defined in the **hosts** parameter of the playbook, which in this case is the Controller group.

CHAPTER 13. CONFIGURING NFS STORAGE

You can configure the overcloud to use shared NFS storage.

13.1. SUPPORTED CONFIGURATIONS AND LIMITATIONS

Supported NFS storage

- Red Hat recommends that you use a certified storage back end and driver. Red Hat does not recommend that you use NFS storage that comes from the generic NFS back end, because its capabilities are limited compared to a certified storage back end and driver. For example, the generic NFS back end does not support features such as volume encryption and volume multi-attach. For information about supported drivers, see the Red Hat Ecosystem Catalog .

- For Block Storage (cinder) and Compute (nova) services, you must use NFS version 4.0 or later. Red Hat OpenStack Platform (RHOSP) does not support earlier versions of NFS.

Unsupported NFS configuration

- RHOSP does not support the NetApp feature NAS secure, because it interferes with normal volume operations. Director disables the feature by default. Therefore, do not edit the following heat parameters that control whether an NFS back end or a NetApp NFS Block Storage back end supports NAS secure:
    - **CinderNetappNasSecureFileOperations**
    - **CinderNetappNasSecureFilePermissions**
    - **CinderNasSecureFileOperations**
    - **CinderNasSecureFilePermissions**

Limitations when using NFS shares

- Instances that have a swap disk cannot be resized or rebuilt when the back end is an NFS share.

13.2. CONFIGURING NFS STORAGE

You can configure the overcloud to use shared NFS storage.

Procedure

1. Create an environment file to configure your NFS storage, for example, **nfs_storage.yaml**.

2. Add the following parameters to your new environment file to configure NFS storage:

```
parameter_defaults:
  CinderEnableIscsiBackend: false
  CinderEnableNfsBackend: true
  GlanceBackend: file
  CinderNfsServers: 192.0.2.230:/cinder
  GlanceNfsEnabled: true
  GlanceNfsShare: 192.0.2.230:/glance
```

NOTE

Do not configure the **CinderNfsMountOptions** and **GlanceNfsOptions** parameters, as their default values enable NFS mount options that are suitable for most Red Hat OpenStack Platform (RHOSP) environments. You can see the value of the **GlanceNfsOptions** parameter in the **environments/storage/glance-nfs.yaml** file. If you experience issues when you configure multiple services to share the same NFS server, contact Red Hat Support.

3. Add your NFS storage environment file to the stack with your other environment files and deploy the overcloud:

```
(undercloud)$ openstack overcloud deploy --templates \
 -e [your environment files] \
 -e /home/stack/templates/nfs_storage.yaml
```

13.3. CONFIGURING AN EXTERNAL NFS SHARE FOR CONVERSION

When the Block Storage service (cinder) performs image format conversion on the overcloud Controller nodes, and the space is limited, conversion of large Image service (glance) images can cause the node root disk space to be completely used. You can use an external NFS share for the conversion to prevent the space on the node from being completely filled.

There are two director heat parameters that control the external NFS share configuration:

- **CinderImageConversionNfsShare**
- **CinderImageConversionNfsOptions**

Procedure

1. Log in to the undercloud as the **stack** user and source the **stackrc** credentials file.

```
$ source ~/stackrc
```

2. In a new or existing storage-related environment file, add information about the external NFS share.

```
parameter_defaults:
  CinderImageConversionNfsShare: 192.168.10.1:/convert
```

NOTE

The default value of the **CinderImageConversionNfsOptions** parameter, that controls the NFS mount options, is sufficient for most environments.

3. Include the environment file that contains your new configuration in the openstack overcloud deploy command with any other environment files that are relevant to your environment.

```
$ openstack overcloud deploy \
--templates \
…
```

```
-e <existing_overcloud_environment_files> \
-e <new_environment_file> \
...
```

- Replace **<existing_overcloud_environment_files>** with the list of environment files that are part of your existing deployment.
- Replace **<new_environment_file>** with the new or edited environment file that contains your NFS share configuration.

CHAPTER 14. 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.

14.1. CHECKING OVERCLOUD DEPLOYMENT STATUS

To check the deployment status of the overcloud, use the **openstack overcloud status** command. This command returns the result of all deployment steps.

Procedure

1. Source the **stackrc** file:

   ```
   $ source ~/stackrc
   ```

2. Run the deployment status command:

   ```
   $ openstack overcloud status
   ```

   The output of this command displays the status of the overcloud:

   ```
   +-----------+---------------------+---------------------+-------------------+
   | Plan Name |       Created       |       Updated       | Deployment Status |
   +-----------+---------------------+---------------------+-------------------+
   | overcloud | 2018-05-03 21:24:50 | 2018-05-03 21:27:59 |   DEPLOY_SUCCESS  |
   +-----------+---------------------+---------------------+-------------------+
   ```

   If your overcloud uses a different name, use the **--stack** argument to select an overcloud with a different name:

   ```
   $ openstack overcloud status --stack <overcloud_name>
   ```

   - Replace **<overcloud_name>** with the name of your overcloud.

14.2. 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.

14.3. 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.

14.4. 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 14.7, "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.

14.5. 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.

14.6. CREATING ADDITIONAL BRIDGE MAPPINGS

Floating IP networks can use any bridge, not just **br-ex**, provided that you map the additional bridge during deployment.

Procedure

1. To map a new bridge called **br-floating** to the **floating** physical network, include the **NeutronBridgeMappings** parameter in an environment file:

```
parameter_defaults:
  NeutronBridgeMappings: "datacentre:br-ex,floating:br-floating"
```

2. With this method, you can create separate external networks after creating the overcloud. For example, to create a floating IP network that maps to the **floating** physical network, run the following commands:

```
$ source ~/overcloudrc
(overcloud) $ openstack network create public --external --provider-physical-network floating
--provider-network-type vlan --provider-segment 105
(overcloud) $ openstack subnet create public --network public --dhcp --allocation-pool
start=10.1.2.51,end=10.1.2.250 --gateway 10.1.2.1 --subnet-range 10.1.2.0/24
```

14.7. 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
```

14.8. 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 15. PERFORMING BASIC OVERCLOUD ADMINISTRATION TASKS

This chapter contains information about basic tasks you might need to perform during the lifecycle of your overcloud.

15.1. ACCESSING OVERCLOUD NODES THROUGH SSH

You can access each overcloud node through the SSH protocol.

- Each overcloud node contains a **tripleo-admin** user.
- The **stack** user on the undercloud has key-based SSH access to the **tripleo-admin** user on each overcloud node.
- All overcloud nodes have a short hostname that the undercloud resolves to an IP address on the control plane network. Each short hostname uses a **.ctlplane** suffix. For example, the short name for **overcloud-controller-0** is **overcloud-controller-0.ctlplane**

Prerequisites

- A deployed overcloud with a working control plane network.

Procedure

1. Log in to the undercloud as the **stack** user.
2. Find the name of the node that you want to access:

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

3. Connect to the node as the **tripleo-admin** user:

```
(undercloud)$ ssh tripleo-admin@overcloud-controller-0
```

15.2. MANAGING CONTAINERIZED SERVICES

Red Hat OpenStack Platform (RHOSP) runs services in containers on the undercloud and overcloud nodes. In certain situations, you might need to control the individual services on a host. This section contains information about some common commands you can run on a node to manage containerized services.

Listing containers and images

To list running containers, run the following command:

```
$ sudo podman ps
```

To include stopped or failed containers in the command output, add the **--all** option to the command:

```
$ sudo podman ps --all
```

To list container images, run the following command:

```
$ sudo podman images
```

Inspecting container properties

To view the properties of a container or container images, use the **podman inspect** command. For example, to inspect the **keystone** container, run the following command:

```
$ sudo podman inspect keystone
```

Managing containers with Systemd services

Previous versions of OpenStack Platform managed containers with Docker and its daemon. Now, the Systemd services interface manages the lifecycle of the containers. Each container is a service and you run Systemd commands to perform specific operations for each container.

NOTE

It is not recommended to use the Podman CLI to stop, start, and restart containers because Systemd applies a restart policy. Use Systemd service commands instead.

To check a container status, run the **systemctl status** command:

```
$ sudo systemctl status tripleo_keystone
● tripleo_keystone.service - keystone container
   Loaded: loaded (/etc/systemd/system/tripleo_keystone.service; enabled; vendor preset: disabled)
   Active: active (running) since Fri 2019-02-15 23:53:18 UTC; 2 days ago
 Main PID: 29012 (podman)
   CGroup: /system.slice/tripleo_keystone.service
           └─29012 /usr/bin/podman start -a keystone
```

To stop a container, run the **systemctl stop** command:

```
$ sudo systemctl stop tripleo_keystone
```

To start a container, run the **systemctl start** command:

```
$ sudo systemctl start tripleo_keystone
```

To restart a container, run the **systemctl restart** command:

```
$ sudo systemctl restart tripleo_keystone
```

Because no daemon monitors the containers status, Systemd automatically restarts most containers in these situations:

- Clean exit code or signal, such as running **podman stop** command.
- Unclean exit code, such as the podman container crashing after a start.
- Unclean signals.
- Timeout if the container takes more than 1m 30s to start.

For more information about Systemd services, see the **systemd.service** documentation.

NOTE

Any changes to the service configuration files within the container revert after restarting the container. This is because the container regenerates the service configuration based on files on the local file system of the node in **/var/lib/config-data/puppet-generated/**. For example, if you edit **/etc/keystone/keystone.conf** within the **keystone** container and restart the container, the container regenerates the configuration using **/var/lib/config-data/puppet-generated/keystone/etc/keystone/keystone.conf** on the local file system of the node, which overwrites any the changes that were made within the container before the restart.

Monitoring podman containers with Systemd timers

The Systemd timers interface manages container health checks. Each container has a timer that runs a service unit that executes health check scripts.

To list all OpenStack Platform containers timers, run the **systemctl list-timers** command and limit the output to lines containing **tripleo**:

```
$ sudo systemctl list-timers | grep tripleo
Mon 2019-02-18 20:18:30 UTC  1s left       Mon 2019-02-18 20:17:26 UTC  1min 2s ago
tripleo_nova_metadata_healthcheck.timer           tripleo_nova_metadata_healthcheck.service
Mon 2019-02-18 20:18:34 UTC  5s left       Mon 2019-02-18 20:17:23 UTC  1min 5s ago
tripleo_keystone_healthcheck.timer                tripleo_keystone_healthcheck.service
Mon 2019-02-18 20:18:35 UTC  6s left       Mon 2019-02-18 20:17:13 UTC  1min 15s ago
tripleo_memcached_healthcheck.timer               tripleo_memcached_healthcheck.service
(...)
```

To check the status of a specific container timer, run the **systemctl status** command for the healthcheck service:

```
$ sudo systemctl status tripleo_keystone_healthcheck.service
● tripleo_keystone_healthcheck.service - keystone healthcheck
   Loaded: loaded (/etc/systemd/system/tripleo_keystone_healthcheck.service; disabled; vendor
preset: disabled)
   Active: inactive (dead) since Mon 2019-02-18 20:22:46 UTC; 22s ago
  Process: 115581 ExecStart=/usr/bin/podman exec keystone /openstack/healthcheck (code=exited,
status=0/SUCCESS)
 Main PID: 115581 (code=exited, status=0/SUCCESS)

Feb 18 20:22:46 undercloud.localdomain systemd[1]: Starting keystone healthcheck...
Feb 18 20:22:46 undercloud.localdomain podman[115581]: {"versions": {"values": [{"status": "stable",
"updated": "2019-01-22T00:00:00Z", "..."}]}]}}
Feb 18 20:22:46 undercloud.localdomain podman[115581]: 300 192.168.24.1:35357 0.012 seconds
Feb 18 20:22:46 undercloud.localdomain systemd[1]: Started keystone healthcheck.
```

To stop, start, restart, and show the status of a container timer, run the relevant **systemctl** command against the **.timer** Systemd resource. For example, to check the status of the **tripleo_keystone_healthcheck.timer** resource, run the following command:

```
$ sudo systemctl status tripleo_keystone_healthcheck.timer
● tripleo_keystone_healthcheck.timer - keystone container healthcheck
   Loaded: loaded (/etc/systemd/system/tripleo_keystone_healthcheck.timer; enabled; vendor preset:
disabled)
   Active: active (waiting) since Fri 2019-02-15 23:53:18 UTC; 2 days ago
```

If the healthcheck service is disabled but the timer for that service is present and enabled, it means that the check is currently timed out, but will be run according to timer. You can also start the check manually.

NOTE

The **podman ps** command does not show the container health status.

Checking container logs

Red Hat OpenStack Platform 17.0 logs all standard output (stdout) from all containers, and standard errors (stderr) consolidated inone single file for each container in **/var/log/containers/stdout**.

The host also applies log rotation to this directory, which prevents huge files and disk space issues.

In case a container is replaced, the new container outputs to the same log file, because **podman** uses the container name instead of container ID.

You can also check the logs for a containerized service with the **podman logs** command. For example, to view the logs for the **keystone** container, run the following command:

```
$ sudo podman logs keystone
```

Accessing containers

To enter the shell for a containerized service, use the **podman exec** command to launch **/bin/bash**. For example, to enter the shell for the **keystone** container, run the following command:

```
$ sudo podman exec -it keystone /bin/bash
```

To enter the shell for the **keystone** container as the root user, run the following command:

```
$ sudo podman exec --user 0 -it <NAME OR ID> /bin/bash
```

To exit the container, run the following command:

```
# exit
```

15.3. MODIFYING THE OVERCLOUD ENVIRONMENT

You can modify the overcloud to add additional features or alter existing operations.

Procedure

1. To modify the overcloud, make modifications to your custom environment files and heat templates, then rerun the **openstack overcloud deploy** command from your initial overcloud creation. For example, if you created an overcloud using Section 11.3, “Configuring and deploying the overcloud”, rerun the following command:

```
$ source ~/stackrc
(undercloud) $ openstack overcloud deploy --templates \
  -e ~/templates/overcloud-baremetal-deployed.yaml \
  -e ~/templates/network-environment.yaml \
  -e ~/templates/storage-environment.yaml \
  --ntp-server pool.ntp.org
```

Director checks the **overcloud** stack in heat, and then updates each item in the stack with the environment files and heat templates. Director does not recreate the overcloud, but rather changes the existing overcloud.

IMPORTANT

Removing parameters from custom environment files does not revert the parameter value to the default configuration. You must identify the default value from the core heat template collection in **/usr/share/openstack-tripleo-heat-templates** and set the value in your custom environment file manually.

2. If you want to include a new environment file, add it to the **openstack overcloud deploy** command with the`-e` option. For example:

```
$ source ~/stackrc
(undercloud) $ openstack overcloud deploy --templates \
  -e ~/templates/new-environment.yaml \
  -e ~/templates/network-environment.yaml \
  -e ~/templates/storage-environment.yaml \
  -e ~/templates/overcloud-baremetal-deployed.yaml \
  --ntp-server pool.ntp.org
```

This command includes the new parameters and resources from the environment file into the stack.

IMPORTANT

It is not advisable to make manual modifications to the overcloud configuration because director might overwrite these modifications later.

15.4. IMPORTING VIRTUAL MACHINES INTO THE OVERCLOUD

You can migrate virtual machines from an existing OpenStack environment to your Red Hat OpenStack Platform (RHOSP) environment.

Procedure

1. On the existing OpenStack environment, create a new image by taking a snapshot of a running server and download the image:

```
$ openstack server image create --name <image_name> <instance_name>
$ openstack image save --file <exported_vm.qcow2> <image_name>
```

- Replace **<instance_name>** with the name of the instance.
- Replace **<image_name>** with the name of the new image.
- Replace **<exported_vm.qcow2>** with the name of the exported virtual machine.

2. Copy the exported image to the undercloud node:

```
$ scp exported_vm.qcow2 stack@192.168.0.2:~/.
```

3. Log in to the undercloud as the **stack** user.
4. Source the **overcloudrc** credentials file:

```
$ source ~/overcloudrc
```

5. Upload the exported image into the overcloud:

```
(overcloud) $ openstack image create --disk-format qcow2  -file <exported_vm.qcow2> --
container-format bare <image_name>
```

6. Launch a new instance:

```
(overcloud) $ openstack server create  --key-name default --flavor m1.demo --image
imported_image --nic net-id=net_id <instance_name>
```

IMPORTANT

You can use these commands to copy each virtual machine disk from the existing OpenStack environment to the new Red Hat OpenStack Platform. QCOW snapshots lose their original layering system.

15.5. LAUNCHING THE EPHEMERAL HEAT PROCESS

In previous versions of Red Hat OpenStack Platform (RHOSP) a system-installed Heat process was used to install the overcloud. Now, we use ephermal Heat to install the overcloud meaning that the **heat-api** and **heat-engine** processes are started on demand by the **deployment**, **update**, and **upgrade** commands.

Previously, you used the **openstack stack** command to create and manage stacks. This command is no longer available by default. For troubleshooting and debugging purposes, for example if the stack should fail, you must first launch the ephemeral Heat process to use the **openstack stack** commands.

Use the **openstack overcloud tripleo launch heat** command to enable ephemeral heat outside of a deployment.

Procedure

1. Use the **openstack tripleo launch heat** command to launch the ephemeral Heat process:

```
(undercloud)$ openstack tripleo launch heat --heat-dir /home/stack/overcloud-
deploy/overcloud/heat-launcher --restore-db
```

The command exits after launching the Heat process, the Heat process continues to run in the background as a podman pod.

2. Use the **podman pod ps** command to verify that the **ephemeral-heat** process is running:

```
(undercloud)$ sudo podman pod ps
POD ID        NAME            STATUS     CREATED        INFRA ID      # OF CONTAINERS
958b141609b2  ephemeral-heat  Running    2 minutes ago  44447995dbcf  3
```

3. Use the **export** command to export the **OS_CLOUD** environment:

```
(undercloud)$ export OS_CLOUD=heat
```

4. Use the **openstack stack list** command to list the installed stacks:

```
(undercloud)$ openstack stack list
+--------------------------------------+------------+---------+-----------------+----------------------+------
--------+
| ID                                   | Stack Name | Project | Stack Status    | Creation Time        |
Updated Time |
+--------------------------------------+------------+---------+-----------------+----------------------+------
--------+
| 761e2a54-c6f9-4e0f-abe6-c8e0ad51a76c | overcloud  | admin   | CREATE_COMPLETE |
2022-08-29T20:48:37Z | None         |
+--------------------------------------+------------+---------+-----------------+----------------------+------
--------+
```

You can debug with commands such as **openstack stack environment show** and **openstack stack resource list**.

5. After you have finished debugging, stop the emphemeral Heat process:

```
(undercloud)$ openstack tripleo launch heat --kill
```

NOTE

Sometimes, exporting the heat environment fails. This can happen when other credentials, such as **overcloudrc**, are in use. In this case unset the existing environment and source the heat environment.

```
(overcloud)$ unset OS_CLOUD
(overcloud)$ unset OS_PROJECT_NAME
(overcloud)$ unset OS_PROJECT_DOMAIN_NAME
(overcloud)$ unset OS_USER_DOMAIN_NAME
(overcloud)$ OS_AUTH_TYPE=none
(overcloud)$ OS_ENDPOINT=http://127.0.0.1:8006/v1/admin
(overcloud)$ export OS_CLOUD=heat
```

15.6. RUNNING THE DYNAMIC INVENTORY SCRIPT

You can run Ansible-based automation in your Red Hat OpenStack Platform (RHOSP) environment. Use the **tripleo-ansible-inventory.yaml** inventory file located in the **/home/stack/overcloud-deploy/<stack>** directory to run ansible plays or ad-hoc commands.

NOTE

If you want to run an Ansible playbook or an Ansible ad-hoc command on the undercloud, you must use the **/home/stack/tripleo-deploy/undercloud/tripleo-ansible-inventory.yaml** inventory file.

Procedure

1. To view your inventory of nodes, run the following Ansible ad-hoc command:

```
(undercloud) [stack@undercloud ~]$ ansible -i ./overcloud-deploy/overcloud/tripleo-ansible-inventory.yaml all --list
```

2. To execute Ansible playbooks on your environment, run the **ansible** command and include the full path to inventory file using the **-i** option. For example:

```
(undercloud) $ ansible <hosts> -i ./overcloud-deploy/tripleo-ansible-inventory.yaml <playbook> <options>
```

- Replace **<hosts>** with the type of hosts that you want to use to use:
  - **controller** for all Controller nodes
  - **compute** for all Compute nodes
  - **overcloud** for all overcloud child nodes. For example, **controller** and **compute** nodes
  - **"*"** for all nodes
- Replace **<options>** with additional Ansible options.
  - Use the **--ssh-extra-args='-o StrictHostKeyChecking=no'** option to bypass confirmation on host key checking.
  - Use the **-u [USER]** option to change the SSH user that executes the Ansible automation. The default SSH user for the overcloud is automatically defined using the **ansible_ssh_user** parameter in the dynamic inventory. The **-u** option overrides this parameter.
  - Use the **-m [MODULE]** option to use a specific Ansible module. The default is **command**, which executes Linux commands.
  - Use the **-a [MODULE_ARGS]** option to define arguments for the chosen module.

IMPORTANT

Custom Ansible automation on the overcloud is not part of the standard overcloud stack. Subsequent execution of the **openstack overcloud deploy** command might override Ansible-based configuration for OpenStack Platform services on overcloud nodes.

15.7. REMOVING AN OVERCLOUD STACK

You can delete an overcloud stack and unprovision all the stack nodes.

NOTE

Deleting your overcloud stack does not erase all the overcloud data. If you need to erase all the overcloud data, contact Red Hat support.

Procedure

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

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

```
$ source ~/stackrc
```

3. Retrieve a list of all the nodes in your stack and their current status:

```
(undercloud)$ openstack baremetal node list
+--------------------------------------+--------------+--------------------------------------+-------------+--
------------------+-------------+
| UUID                                 | Name         | Instance UUID                        | Power State |
Provisioning State | Maintenance |
+--------------------------------------+--------------+--------------------------------------+-------------+--
------------------+-------------+
| 92ae71b0-3c31-4ebb-b467-6b5f6b0caac7 | compute-0    | 059fb1a1-53ea-4060-9a47-
09813de28ea1 | power on    | active             | False       |
| 9d6f955e-3d98-4d1a-9611-468761cebabf | compute-1    | e73a4b50-9579-4fe1-bd1a-
556a2c8b504f | power on    | active             | False       |
| 8a686fc1-1381-4238-9bf3-3fb16eaec6ab | controller-0 | 6d69e48d-10b4-45dd-9776-
155a9b8ad575 | power on    | active             | False       |
| eb8083cc-5f8f-405f-9b0c-14b772ce4534 | controller-1 | 1f836ac0-a70d-4025-88a3-
bbe0583b4b8e | power on    | active             | False       |
| a6750f1f-8901-41d6-b9f1-f5d6a10a76c7 | controller-2 | e2edd028-cea6-4a98-955e-
5c392d91ed46 | power on    | active             | False       |
+--------------------------------------+--------------+--------------------------------------+-------------+--
------------------+-------------+
```

4. Delete the overcloud stack and unprovision the nodes and networks:

```
(undercloud)$ openstack overcloud delete -b <node_definition_file> \
 --networks-file <networks_definition_file> --network-ports <stack>
```

- Replace **<node_definition_file>** with the name of your node definition file, for example, **overcloud-baremetal-deploy.yaml**.
- Replace **<networks_definition_file>** with the name of your networks definition file, for example, **network_data_v2.yaml**.
- Replace **<stack>** with the name of the stack that you want to delete. If not specified, the default stack is **overcloud**.

5. Confirm that you want to delete the overcloud:

```
Are you sure you want to delete this overcloud [y/N]?
```

6. Wait for the overcloud to delete and the nodes and networks to unprovision.

7. Confirm that the bare-metal nodes have been unprovisioned:

```
(undercloud) [stack@undercloud-0 ~]$ openstack baremetal node list
+--------------------------------------+--------------+---------------+-------------+--------------------+----
---------+
| UUID                                 | Name         | Instance UUID | Power State | Provisioning State |
Maintenance |
+--------------------------------------+--------------+---------------+-------------+--------------------+----
---------+
| 92ae71b0-3c31-4ebb-b467-6b5f6b0caac7 | compute-0    | None          | power off   |
available          | False       |
| 9d6f955e-3d98-4d1a-9611-468761cebabf | compute-1    | None          | power off   |
available          | False       |
| 8a686fc1-1381-4238-9bf3-3fb16eaec6ab | controller-0 | None          | power off   | available
| False       |
| eb8083cc-5f8f-405f-9b0c-14b772ce4534 | controller-1 | None          | power off   | available
| False       |
| a6750f1f-8901-41d6-b9f1-f5d6a10a76c7 | controller-2 | None          | power off   | available
| False       |
+--------------------------------------+--------------+---------------+-------------+--------------------+----
---------+
```

8. Remove the stack directories:

```
$ rm -rf ~/overcloud-deploy/<stack>
$ rm -rf ~/config-download/<stack>
```

NOTE

The directory paths for your stack might be different from the default if you used the **--output-dir** and **--working-dir** options when deploying the overcloud with the **openstack overcloud deploy** command.

CHAPTER 16. 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.

16.1. 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) 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.

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**.

16.2. CONFIG-DOWNLOAD WORKING DIRECTORY

The **ansible-playbook** command creates an Ansible project directory, default name ~/**config-download/overcloud**. This project directory stores downloaded software configuration from heat. It includes all Ansible-related files which you need to run **ansible-playbook** to configure the overcloud.

The contents of the directory include:

- tripleo-ansible-inventory.yaml - Ansible inventory file containing **hosts** and **vars** for all the overcloud nodes.

- ansible.log - Log file from the most recent run of **ansible-playbook**.
- ansible.cfg - Configuration file used when running **ansible-playbook**.
- ansible-playbook-command.sh - Executable script used to rerun **ansible-playbook**.
- ssh_private_key - Private ssh key used to access the overcloud nodes.

   1. Reproducing ansible-playbook

After the project directory is created, run the **ansible-playbook-command.sh** command to reproduce the deployment.

```
$ ./ansible-playbook-command.sh
```

You can run the script with additional arguments, such as check mode **--check**, limiting hosts **--limit**, and overriding variables **-e**.

```
$ ./ansible-playbook-command.sh --check
```

16.3. CHECKING CONFIG-DOWNLOAD LOG

During the **config-download** process, Ansible creates a log file, named **ansible.log**, in the **/home/stack** directory on the undercloud.

Procedure

1. View the log with the **less** command:

   ```
   $ less ~/ansible.log
   ```

16.4. 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, usually named **overcloud**:

   ```
   $ cd ~/config-download/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
   ```

16.5. 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.

16.6. 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 **~/config-download/overcloud** 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.

16.7. 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 **~/config-download/overcloud** 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.

16.8. 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 **~/config-download/overcloud**. 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. Run the **ansible-playbook-command.sh** script.
   You can pass additional Ansible arguments to this script, which are then passed unchanged to the **ansible-playbook** command. This makes it possible to take advantage of 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.

   NOTE

   To ensure that the **/etc/hosts** file, on all nodes, is up-to-date, run the following command as the **stack** user:

   ```
   (undercloud)$ cd /home/stack/overcloud-deploy/overcloud/config-download/overcloud
   (undercloud)$ ANSIBLE_REMOTE_USER="tripleo-admin" ansible allovercloud \
     -i /home/stack/overcloud-deploy/overcloud/tripleo-ansible-inventory.yaml \
     -m include_role \
     -a name=tripleo_hosts_entries \
     -e @global_vars.yaml
   ```

3. 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
  ```

4. 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"
   ```

16.9. 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 deploy \
  --stack overcloud --stack-only \
  --config-dir  ~/overcloud-deploy/overcloud/config-download/overcloud/
```

- **--stack** specifies the name of the overcloud.
- **--stack-only** ensures that the command only deploys the heat stack and skips any software configuration.
- **--config-dir** specifies the location of 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 tripleo-admin \
  --static-yaml-inventory inventory.yaml
```

- Replace **<overcloud>** with the name of your overcloud.

9. Use the **~/overcloud-deploy/overcloud/config-download/overcloud** 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 \
  ~/overcloud-deploy/overcloud/config-download/overcloud/deploy_steps_playbook.yaml
```

NOTE

When you run the **config-download/overcloud** 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"** in 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 \
  ~/overcloud-deploy/overcloud/config-
download/overcloud/deploy_steps_playbook.yaml
```

10. Wait until the configuration process completes.

11. Generate an **overcloudrc** file manually from the ansible-based configuration:

```
$ 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.

NOTE

The ~/**overcloud-deploy**/**overcloud**/**config-download**/**overcloud**/ directory contains a playbook called **deploy_steps_playbook.yaml**. The playbook uses various task files contained in the working directory. Some task files are common to all Red Hat OpenStack Platform (RHOSP) roles and some are specific to certain RHOSP roles and servers.

The ~/**overcloud-deploy**/**overcloud**/**config-download**/**overcloud**/ directory also contains sub-directories that correspond to each role that you define in your overcloud **roles_data** file. Each RHOSP role directory also contains sub-directories for individual servers of that role type. The directories use the composable role hostname format, for example **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
```

You can 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 \
  ~/overcloud-deploy/overcloud/config-
download/overcloud/deploy_steps_playbook.yaml
```

16.10. 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.

16.11. 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.

16.12. 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.

16.13. 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 **tripleo-container-manage** (container configuration and management).
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 17. MANAGING CONTAINERS WITH ANSIBLE

Red Hat OpenStack Platform 17.0 uses the **tripleo_container_manage** Ansible role to perform management operations on containers. 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.

17.1. TRIPLEO-CONTAINER-MANAGE ROLE DEFAULTS AND VARIABLES

The following excerpt shows the defaults and variables for the **tripleo_container_manage** Ansible role.

```
# All variables intended for modification should place placed in this file.
tripleo_container_manage_hide_sensitive_logs: '{{ hide_sensitive_logs | default(true)
  }}'
tripleo_container_manage_debug: '{{ ((ansible_verbosity | int) >= 2) | bool }}'
tripleo_container_manage_clean_orphans: true

# All variables within this role should have a prefix of "tripleo_container_manage"
tripleo_container_manage_check_puppet_config: false
tripleo_container_manage_cli: podman
tripleo_container_manage_concurrency: 1
tripleo_container_manage_config: /var/lib/tripleo-config/
tripleo_container_manage_config_id: tripleo
tripleo_container_manage_config_overrides: {}
tripleo_container_manage_config_patterns: '*.json'
# Some containers where Puppet is run, can take up to 10 minutes to finish
# in slow environments.
tripleo_container_manage_create_retries: 120
# Default delay is 5s so 120 retries makes a timeout of 10 minutes which is
# what we have observed a necessary value for nova and neutron db-sync execs.
tripleo_container_manage_exec_retries: 120
tripleo_container_manage_healthcheck_disabled: false
tripleo_container_manage_log_path: /var/log/containers/stdouts
tripleo_container_manage_systemd_teardown: true
```

17.2. TRIPLEO-CONTAINER-MANAGE MOLECULE SCENARIOS

Molecule is used to test the **tripleo_container_manage** role. The following shows a **molecule** default inventory:

```
hosts:
  all:
    hosts:
      instance:
        ansible_host: localhost
        ansible_connection: local
        ansible_distribution: centos8
```

Usage

Red Hat OpenStack 17.0 supports only Podman in this role. Docker support is on the roadmap.

The **Molecule** Ansible role performs the following tasks:

- Collect container configuration data, generated by the TripleO Heat Templates. This data is used as a source of truth. If a container is already managed by **Molecule**, no matter its present state, the configuration data will reconfigure the container as needed.

- Manage **systemd** shutdown files. It creates the **TripleO Container systemd** service, required for service ordering when shutting down or starting a node. It also manages the **netns-placeholder** service.

- Delete containers that are nonger needed or that require reconfiguration. It uses a custom filter, named **needs_delete()** which has a set of rules to determine if the container needs to be deleted.

  - A container will not be deleted if, the container is not managed by **tripleo_ansible** or the container **config_id** does not match the input ID.

  - A container will be deleted, if the container has no **config_data** or the container has **config_data** which does not match data in input. Note that when a container is removed, the role also disables and removes the **systemd** services and healtchecks.

- Create containers in a specific order defined by **start_order** container config, where the default is 0.

  - If the container is an **exec**, a dedicated playbook for **execs** is run, using **async** so multiple **execs** can be run at the same time.

  - Otherwise, the **podman_container** is used, in async, to create the containers. If the container has a **restart policy**, **systemd** service is configured. If the container has a healthcheck script, **systemd healthcheck** service is configured.

> NOTE
>
> **tripleo_container_manage_concurrency** parameter is set to 1 by default, and putting a value higher than 2 can expose issues with Podman locks.

Example of a playbook:

```
- name: Manage step_1 containers using tripleo-ansible
```

```
block:
  - name: "Manage containers for step 1 with tripleo-ansible"
    include_role:
      name: tripleo_container_manage
    vars:
      tripleo_container_manage_config: "/var/lib/tripleo-config/container-startup-config/step_1"
      tripleo_container_manage_config_id: "tripleo_step1"
```

17.3. TRIPLEO_CONTAINER_MANAGE ROLE VARIABLES

The **tripleo_container_manage** Ansible role contains the following variables:

Table 17.1. 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.
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.

Name	Default value	Description
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.
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]**.

17.4. TRIPLEO-CONTAINER-MANAGE HEALTHCHECKS

Until Red Hat OpenStack 17.0, container healthcheck was implemented by a **systemd** timer which would run **podman exec** to determine if a given container was healthy. Now, it uses the native healthcheck interface in **Podman** which is easier to integrate and consume.

To check if a container (for example, keystone) is healthy, run the following command:

```
$ sudo podman healthcheck run keystone
```

The return code should be **0** and **"healthy"**.

```
"Healthcheck": {
    "Status": "healthy",
    "FailingStreak": 0,
    "Log": [
        {
            "Start": "2020-04-14T18:48:57.272180578Z",
            "End": "2020-04-14T18:48:57.806659104Z",
            "ExitCode": 0,
            "Output": ""
        },
        (...)
    ]
}
```

17.5. TRIPLEO-CONTAINER-MANAGE DEBUG

The **tripleo_container_manage** Ansible role allows you to perform specific actions on a given container. This can be used to:

- Run a container with a specific one-off configuration.
- Output the container commands to manage containers lifecycle.
- Output the changes made on containers by Ansible.

NOTE

To manage a single container, you need to know two things:

- At which step during the overcloud deployment was the container deployed.
- The name of the generated JSON file containing the container configuration.

The following is an example of a playbook to manage **HAproxy** container at **step 1** which overrides the image setting:

```
- 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_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_clean_orphans: false
          tripleo_container_manage_config_overrides:
            haproxy:
              image: quay.io/tripleomastercentos9/centos-binary-haproxy:hotfix
```

If Ansible is run in **check mode**, no container is removed or created, however at the end of the playbook run a list of commands is displayed to show the possible outcome of the playbook. This is useful for debugging purposes.

```
$ ansible-playbook haproxy.yaml --check
```

Adding the **diff mode** will show the changes that would have been made on containers by Ansible.

```
$ ansible-playbook haproxy.yaml --check --diff
```

The **tripleo_container_manage_clean_orphans** parameter is optional. It can be set to false meaning orphaned containers, with a specific **config_id**, will not be removed. It can be used to manage a single container without impacting other running containers with same **config_id**.

The **tripleo_container_manage_config_overrides** parameter is optional and can be used to override a specific container attribute, for example the image or the container user. The parameter creates dictionary with container name and the parameters to override. These parameters have to exist and they define the container configuration in TripleO Heat Templates.

Note the dictionary does not update the overrides in the JSON file so if an update or upgrade is executed, the container will be re-configured with the configuration in the JSON file.

CHAPTER 18. USING THE VALIDATION FRAMEWORK

Red Hat OpenStack Platform (RHOSP) 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 **validation** 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 **validation run** command to refine the validation according to your environment.

18.1. ANSIBLE-BASED VALIDATIONS

During the installation of Red Hat OpenStack Platform (RHOSP) 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 RHOSP deployment before an upgrade.

post-upgrade

Validations to validate your RHOSP deployment after an upgrade.

18.2. CHANGING THE VALIDATION CONFIGURATION FILE

The validation configuration file is a .ini file that you can edit to control every aspect of the validation execution and the communication between remote machines.

You can change the default configuration values in one of the following ways:

- Edit the default /etc/validations.cfg file.

- Make your own copy of the default **/etc/validations.cfg** file, edit the copy, and provide it through the CLI with the **--config** argument. If you create your own copy of the configuration file, point the CLI to this file on each execution with **--config**.

By default, the location of the validation configuration file is **/etc/validation.cfg**.

IMPORTANT

Ensure that you correctly edit the configuration file or your validation might fail with errors, for example:

- undetected validations
- callbacks written to different locations
- incorrectly-parsed logs

Prerequisites

- You have a thorough understanding of how to validate your environment.

Procedure

1. Optional: Make a copy of the validation configuration file for editing:
   a. Copy **/etc/validation.cfg** to your home directory.
   b. Make the required edits to the new configuration file.
2. Run the validation command:

```
$ validation run --config <configuration-file>
```

   - Replace <configuration-file> with the file path to the configuration file that you want to use.

NOTE

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

18.3. LISTING VALIDATIONS

Run the **validation list** command to list the different types of validations available.

Procedure

1. Source the **stackrc** file.

```
$ source ~/stackrc
```

2. Run the **validation list** command:

   - To list all validations, run the command without any options:

```
$ validation list
```

- To list validations in a group, run the command with the **--group** option:

  ```
  $ validation list --group prep
  ```

> **NOTE**
>
> For a full list of options, run **validation list --help**.

18.4. RUNNING VALIDATIONS

To run a validation or validation group, use the **validation run** command. To see a full list of options, use the **validation run --help** command.

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

Procedure

1. Source the **stackrc** file:

   ```
   $ source ~/stackrc
   ```

2. Validate a static inventory file called **tripleo-ansible-inventory.yaml**.

   ```
   $ validation run --group pre-introspection -i tripleo-ansible-inventory.yaml
   ```

   > **NOTE**
   >
   > You can find the inventory file in the **~/tripleo-deploy/<stack>** directory for a standalone or undercloud deployment or in the **~/overcloud-deploy/<stack>** directory for an overcloud deployment.

3. Enter the **validation 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 memory requirements of each node, enter **--validation check-ram**:

     ```
     $ validation run --validation check-ram
     ```

     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:

     ```
     $ validation run --group prep
     ```

To view detailed output from a specific validation, run the **validation history get --full** command against the UUID of the specific validation from the report:

```
$ validation history get --full <UUID>
```

18.5. CREATING A VALIDATION

You can create a validation with the **validation init** command. Execution of the command results in a basic template for a new validation. You can edit the new validation role to suit your requirements.

IMPORTANT

Red Hat does not support user-created validations.

Prerequisites

- You have a thorough understanding of how to validate your environment.
- You have access rights to the directory where you run the command.

Procedure

1. Create your validation:

   ```
   $ validation init <my-new-validation>
   ```

   - Replace **<my-new-validation>** with the name of your new validation.
     The execution of this command results in the creation of the following directory and sub-directories:

     ```
     /home/stack/community-validations
     ├── library
     ├── lookup_plugins
     ├── playbooks
     └── roles
     ```

     NOTE

     If you see the error message "The Community Validations are disabled by default, ensure that the **enable_community_validations** parameter is set to **True** in the validation configuration file. The default name and location of this file is **/etc/validation.cfg**.

2. Edit the role to suit your requirements.

Additional resources

- Section 18.2, “Changing the validation configuration file”.

18.6. 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 **validation history list** command.

Prerequisites

- You have run a validation or group of validations.

Procedure

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

   ```
   $ source ~/stackrc
   ```

3. You can view a list of all validations or the most recent validations:
   - View a list of all validations:

     ```
     $ validation history list
     ```

   - View history for a specific validation type by using the **--validation** option:

     ```
     $ validation history get --validation <validation-type>
     ```

     - Replace *<validation-type>* with the type of validation, for example, ntp.
4. View the log for a specific validation UUID:

   ```
   $ validation show run --full 7380fed4-2ea1-44a1-ab71-aab561b44395
   ```

Additional resources

- Using the validation framework

18.7. 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 **validation history get --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.

18.8. VALIDATION FRAMEWORK LOG OUTPUT FORMATS

The default behaviour of the validation framework is to save validation logs in JSON format. You can change the output of the logs with the **ANSIBLE_STDOUT_CALLBACK** environment variable.

To change the validation output log format, run a validation and include the **--extra-env-vars ANSIBLE_STDOUT_CALLBACK=<callback>** option:

```
$ validation run --extra-env-vars ANSIBLE_STDOUT_CALLBACK=<callback> --validation check-ram
```

- Replace **<callback>** with an Ansible output callback. To view a list of the standard Ansible output callbacks, run the following command:

```
$ ansible-doc -t callback -l
```

The validation framework includes the following additional callbacks:

validation_json
: The framework saves JSON-formatted validation results as a log file in **/var/logs/validations**. This is the default callback for the validation framework.

validation_stdout

The framework displays JSON-formatted validation results on screen.

http_json

The framework sends JSON-formatted validation results to an external logging server. You must also include additional environment variables for this callback:

HTTP_JSON_SERVER

The URL for the external server.

HTTP_JSON_PORT

The port for the API entry point of the external server. The default port in 8989.

Set these environment variables with additional **--extra-env-vars** options:

```
$ validation run --extra-env-vars ANSIBLE_STDOUT_CALLBACK=http_json \
    --extra-env-vars HTTP_JSON_SERVER=http://logserver.example.com \
    --extra-env-vars HTTP_JSON_PORT=8989 \
    --validation check-ram
```

IMPORTANT

Before you use the **http_json** callback, you must add **http_json** to the **callback_whitelist** parameter in your **ansible.cfg** file:

```
callback_whitelist = http_json
```

18.9. IN-FLIGHT VALIDATIONS

Red Hat OpenStack Platform (RHOSP) 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 19. SCALING OVERCLOUD NODES

WARNING

The content for this feature is available in this release as a *Documentation Preview*, and therefore is not fully verified by Red Hat. Use it only for testing, and do not use in a production environment.

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 19.1. 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 20, *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.

19.1. ADDING NODES TO THE OVERCLOUD

You can add more nodes to your overcloud.

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
(undercloud)$ openstack overcloud node import newnodes.json
```

3. Launch the introspection process for each new node:

```
(undercloud)$ openstack overcloud node introspect \
 --provide <node_1> [node_2] [node_n]
```

   - Use the **--provide** option to reset all the specified nodes to an **available** state after introspection.
   - Replace **<node_1>**, **[node_2]**, and all nodes up to **[node_n]** with the UUID of each node that you want to introspect.

4. Configure the image properties for each new node:

```
(undercloud)$ openstack overcloud node configure <node>
```

19.2. SCALING UP BARE-METAL NODES

To increase the count of bare-metal nodes in an existing overcloud, increment the node count in the **overcloud-baremetal-deploy.yaml** file and redeploy the overcloud.

Prerequisites

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

Procedure

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

```
$ source ~/stackrc
```

2. Open the **overcloud-baremetal-deploy.yaml** node definition file that you use to provision your bare-metal nodes.

3. Increment the **count** parameter for the roles that you want to scale up. For example, the following configuration increases the Object Storage node count to 4:

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

```
  count: 10
- name: ObjectStorage
  count: 4
```

4. Optional: Configure predictive node placement for the new nodes. For example, use the following configuration to provision a new Object Storage node on **node03**:

```
- name: ObjectStorage
  count: 4
  instances:
  - hostname: overcloud-objectstorage-0
    name: node00
  - hostname: overcloud-objectstorage-1
    name: node01
  - hostname: overcloud-objectstorage-2
    name: node02
  - hostname: overcloud-objectstorage-3
    name: node03
```

5. Optional: Define any other attributes that you want to assign to your new nodes. For more information about the properties you can use to configure node attributes in your node definition file, see Bare-metal node provisioning attributes.

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

7. Provision the overcloud nodes:

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

   - Replace **<stack>** with the name of the stack for which the bare-metal nodes are provisioned. If not specified, the default is **overcloud**.
   - Replace **<deployment_file>** with the name of the heat environment file to generate for inclusion in the deployment command, for example **/home/stack/templates/overcloud-baremetal-deployed.yaml**.

8. Monitor the provisioning progress in a separate terminal. When provisioning is successful, the node state changes from **available** to **active**:

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

9. Add the generated **overcloud-baremetal-deployed.yaml** file to the stack with your other environment files and deploy the overcloud:

```
(undercloud)$ openstack overcloud deploy --templates \
  -e [your environment files] \
  -e /home/stack/templates/overcloud-baremetal-deployed.yaml \
--deployed-server \
--disable-validations \
...
```

19.3. SCALING DOWN BARE-METAL NODES

To scale down the number of bare-metal nodes in your overcloud, tag the nodes that you want to delete from the stack in the node definition file, redeploy the overcloud, and then delete the bare-metal node from the overcloud.

Prerequisites

- A successful undercloud installation. For more information, see Installing director on the undercloud.
- A successful overcloud deployment. For more information, see Configuring a basic overcloud with pre-provisioned nodes.
- If you are replacing an Object Storage node, replicate data from the node you are removing to the new replacement node. Wait for a replication pass to finish on the new node. Check the replication pass progress in the **/var/log/swift/swift.log** file. When the pass finishes, the Object Storage service (swift) adds entries to the log similar to the following example:

  ```
  Mar 29 08:49:05 localhost object-server: Object replication complete.
  Mar 29 08:49:11 localhost container-server: Replication run OVER
  Mar 29 08:49:13 localhost account-server: Replication run OVER
  ```

Procedure

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

   ```
   $ source ~/stackrc
   ```

2. Decrement the **count** parameter in the **overcloud-baremetal-deploy.yaml** file, for the roles that you want to scale down.
3. Define the **hostname** and **name** of each node that you want to remove from the stack, if they are not already defined in the **instances** attribute for the role.
4. Add the attribute **provisioned: false** to the node that you want to remove. For example, to remove the node **overcloud-objectstorage-1** from the stack, include the following snippet in your **overcloud-baremetal-deploy.yaml** file:

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

```
  name: node02
- hostname: overcloud-objectstorage-3
  name: node03
```

After you redeploy the overcloud, the nodes that you define with the **provisioned: false** attribute are no longer present in the stack. However, these nodes are still running in a provisioned state.

NOTE

To remove a node from the stack temporarily, deploy the overcloud with the attribute **provisioned: false** and then redeploy the overcloud with the attribute **provisioned: true** to return the node to the stack.

5. Delete the node from the overcloud:

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

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

NOTE

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

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

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

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

7. Add the **overcloud-baremetal-deployed.yaml** file generated by the provisioning command to the stack with your other environment files, and deploy the overcloud:

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

19.4. REMOVING OR REPLACING A COMPUTE NODE

In some situations you need to remove a Compute node from the overcloud. For example, you might need to replace a problematic Compute node. When you delete a Compute node the node's index is added by default to the denylist to prevent the index being reused during scale out operations.

You can replace the 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:

  ```
  (overcloud)$ openstack compute service list
  ```

  If the Compute service is not disabled then disable it:

  ```
  (overcloud)$ openstack compute service set <hostname> nova-compute --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:

    ```
    [root@controller-0 ~]# pcs stonith disable <stonith_resource_name>
    [stack@undercloud ~]$ source stackrc
    [stack@undercloud ~]$ openstack baremetal node power off <UUID>
    ```

Procedure

1. Source the undercloud configuration:

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

2. Decrement the **count** parameter in the **overcloud-baremetal-deploy.yaml** file, for the roles that you want to scale down.
3. Define the **hostname** and **name** of each node that you want to remove from the stack, if they are not already defined in the **instances** attribute for the role.
4. Add the attribute **provisioned: false** to the node that you want to remove. For example, to remove the node **overcloud-compute-1** from the stack, include the following snippet in your **overcloud-baremetal-deploy.yaml** file:

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

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. Delete the node from the overcloud:

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

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

NOTE

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

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

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

- Replace **<stack>** with the name of the stack for which the bare-metal nodes are provisioned. If not specified, the default is **overcloud**.
- Replace **<deployment_file>** with the name of the heat environment file to generate for inclusion in the deployment command, for example **/home/stack/templates/overcloud-baremetal-deployed.yaml**.

7. 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
```

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

b. Delete the STONITH device for the node:

```
$ sudo pcs stonith delete <device-name>
```

8. If you are replacing the removed Compute node on your overcloud deployment, see Replacing a removed Compute node.

19.4.1. Removing a Compute node manually

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. Use the **openstack tripleo launch heat** command to launch the ephemeral Heat process:

```
(undercloud)$ openstack tripleo launch heat --heat-dir /home/stack/overcloud-
deploy/overcloud/heat-launcher --restore-db
```

   The command exits after launching the Heat process, the Heat process continues to run in the background as a podman pod.

2. Use the **podman pod ps** command to verify that the **ephemeral-heat** process is running:

```
(undercloud)$ sudo podman pod ps
POD ID       NAME            STATUS      CREATED          INFRA ID       # OF CONTAINERS
958b141609b2  ephemeral-heat  Running     2 minutes ago  44447995dbcf  3
```

3. Use the **export** command to export the **OS_CLOUD** environment:

```
(undercloud)$ export OS_CLOUD=heat
```

4. Use the **openstack stack list** command to list the installed stacks:

```
(undercloud)$ openstack stack list
+--------------------------------------+------------+---------+-----------------+----------------------+------
--------+
| ID                                   | Stack Name | Project | Stack Status    | Creation Time        |
Updated Time |
+--------------------------------------+------------+---------+-----------------+----------------------+------
--------+
| 761e2a54-c6f9-4e0f-abe6-c8e0ad51a76c | overcloud  | admin   | CREATE_COMPLETE |
2022-08-29T20:48:37Z | None         |
+--------------------------------------+------------+---------+-----------------+----------------------+------
--------+
```

5. Identify the UUID of the node that you want to manually delete:

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

6. Move the node that you want to delete to maintenance mode:

```
(undercloud)$ openstack baremetal node maintenance set <node_uuid>
```

7. Wait for the Compute service to synchronize its state with the Bare Metal service. This can take up to four minutes.

8. Source the overcloud configuration:

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

9. 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 to remove.

10. 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
```

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.

11. Remove the deleted Compute service as a resource provider from the Placement service:

```
(overcloud)$ openstack resource provider list
(overcloud)$ openstack resource provider delete <uuid>
```

12. Source the undercloud configuration:

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

13. 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 host name 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 `~/ansible.log`
> WARNING: Scale-down configuration error. Manual cleanup of some
> actions may be necessary. Continuing with node removal.
> ```
>
> You can ignore this message.

14. Wait for the overcloud node to delete.

15. Check the status of the overcloud stack when the node deletion is complete:

```
(undercloud)$ openstack stack list
```

Table 19.2. Result

Status	Description
UPDATE_COMPLETE	The delete operation completed successfully.
UPDATE_FAILED	The delete operation failed. If the overcloud node fails to delete while in maintenance mode, then the problem might be with the hardware.

16. 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>
```

17. 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 **overcloud-baremetal-deployed.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.

19.4.2. 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 and deploying the overcloud.
    - To re-add a Compute node that you removed manually, remove the node from maintenance mode:

      ```
      (undercloud)$ 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:

    ```
    (undercloud)$ 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.

19.5. REPLACING CEPH STORAGE NODES

You can use director to replace Ceph Storage nodes in a director-created cluster. For more information, see the Deploying Red Hat Ceph Storage and Red Hat OpenStack Platform together with director guide.

19.6. 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.

19.7. BLACKLISTING NODES

You can exclude overcloud nodes from receiving an updated deployment. This is useful in scenarios where you want to scale new nodes and exclude existing nodes from receiving an updated set of parameters and resources from the core heat template collection. This means that the blacklisted nodes are isolated from the effects of the stack operation.

Use the **DeploymentServerBlacklist** parameter in an environment file to create a blacklist.

Setting the blacklist

The **DeploymentServerBlacklist** parameter is a list of server names. Write a new environment file, or add the parameter value to an existing custom environment file and pass the file to the deployment command:

```
parameter_defaults:
  DeploymentServerBlacklist:
    - overcloud-compute-0
```

```
    - overcloud-compute-1
    - overcloud-compute-2
```

NOTE

The server names in the parameter value are the names according to OpenStack Orchestration (heat), not the actual server hostnames.

Include this environment file with your **openstack overcloud deploy** command:

```
$ source ~/stackrc
(undercloud) $ openstack overcloud deploy --templates \
  -e server-blacklist.yaml \
  [OTHER OPTIONS]
```

Heat blacklists any servers in the list from receiving updated heat deployments. After the stack operation completes, any blacklisted servers remain unchanged. You can also power off or stop the **os-collect-config** agents during the operation.

WARNING

- Exercise caution when you blacklist nodes. Only use a blacklist if you fully understand how to apply the requested change with a blacklist in effect. It is possible to create a hung stack or configure the overcloud incorrectly when you use the blacklist feature. For example, if cluster configuration changes apply to all members of a Pacemaker cluster, blacklisting a Pacemaker cluster member during this change can cause the cluster to fail.

- Do not use the blacklist during update or upgrade procedures. Those procedures have their own methods for isolating changes to particular servers.

- When you add servers to the blacklist, further changes to those nodes are not supported until you remove the server from the blacklist. This includes updates, upgrades, scale up, scale down, and node replacement. For example, when you blacklist existing Compute nodes while scaling out the overcloud with new Compute nodes, the blacklisted nodes miss the information added to **/etc/hosts** and **/etc/ssh/ssh_known_hosts**. This can cause live migration to fail, depending on the destination host. The Compute nodes are updated with the information added to **/etc/hosts** and **/etc/ssh/ssh_known_hosts** during the next overcloud deployment where they are no longer blacklisted. Do not modify the **/etc/hosts** and **/etc/ssh/ssh_known_hosts** files manually. To modify the **/etc/hosts** and **/etc/ssh/ssh_known_hosts** files, run the overcloud deploy command as described in the *Clearing the Blacklist* section.

Clearing the blacklist

To clear the blacklist for subsequent stack operations, edit the **DeploymentServerBlacklist** to use an empty array:

```
parameter_defaults:
  DeploymentServerBlacklist: []
```

WARNING

Do not omit the **DeploymentServerBlacklist** parameter. If you omit the parameter, the overcloud deployment uses the previously saved value.

CHAPTER 20. 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.

20.1. PREPARING FOR CONTROLLER REPLACEMENT

Before you replace an overcloud Controller node, it is important to check the current state of your Red Hat OpenStack Platform environment. Checking the current state can help avoid complications during the Controller replacement process. Use the following list of preliminary checks to determine if it is safe to perform a Controller node replacement. Run all commands for these checks on the undercloud.

Procedure

1. Check the current status of the **overcloud** stack on the undercloud:

   ```
   $ source stackrc
   $ openstack overcloud status
   ```

   Only continue if the **overcloud** stack has a deployment status of **DEPLOY_SUCCESS**.

2. Install the database client tools:

   ```
   $ sudo dnf -y install mariadb
   ```

3. Configure root user access to the database:

   ```
   $ sudo cp /var/lib/config-data/puppet-generated/mysql/root/.my.cnf /root/.
   ```

4. Perform a backup of the undercloud databases:

   ```
   $ mkdir /home/stack/backup
   $ sudo mysqldump --all-databases --quick --single-transaction | gzip >
   /home/stack/backup/dump_db_undercloud.sql.gz
   ```

5. Check that your undercloud contains 10 GB free storage to accommodate for image caching and conversion when you provision the new node:

   ```
   $ df -h
   ```

6. If you are reusing the IP address for the new controller node, ensure that you delete the port used by the old controller:

   ```
   $ openstack port delete <port>
   ```

7. Check the status of Pacemaker on the running Controller nodes. For example, if 192.168.0.47 is the IP address of a running Controller node, use the following command to view the Pacemaker status:

   ```
   $ ssh tripleo-admin@192.168.0.47 'sudo pcs status'
   ```

   The output shows all services that are running on the existing nodes and those that are stopped on the failed node.

8. Check the following parameters on each node of the overcloud MariaDB cluster:

   - **wsrep_local_state_comment: Synced**
   - **wsrep_cluster_size: 2**
     Use the following command to check these parameters on each running Controller node. In this example, the Controller node IP addresses are 192.168.0.47 and 192.168.0.46:

     ```
     $ for i in 192.168.0.46 192.168.0.47 ; do echo "*** $i ***" ; ssh tripleo-admin@$i "sudo
     podman exec \$(sudo podman ps --filter name=galera-bundle -q) mysql -e \"SHOW
     STATUS LIKE 'wsrep_local_state_comment'; SHOW STATUS LIKE
     'wsrep_cluster_size';\""; done
     ```

9. Check the RabbitMQ status. For example, if 192.168.0.47 is the IP address of a running Controller node, use the following command to view the RabbitMQ status:

   ```
   $ ssh tripleo-admin@192.168.0.47 "sudo podman exec \$(sudo podman ps -f
   name=rabbitmq-bundle -q) rabbitmqctl cluster_status"
   ```

   The **running_nodes** key should show only the two available nodes and not the failed node.

10. If fencing is enabled, disable it. For example, if 192.168.0.47 is the IP address of a running Controller node, use the following command to check the status of fencing:

    ```
    $ ssh tripleo-admin@192.168.0.47 "sudo pcs property show stonith-enabled"
    ```

    Run the following command to disable fencing:

    ```
    $ ssh tripleo-admin@192.168.0.47 "sudo pcs property set stonith-enabled=false"
    ```

11. Login to the failed Controller node and stop all the **nova_*** containers that are running:

    ```
    $ sudo systemctl stop tripleo_nova_api.service
    $ sudo systemctl stop tripleo_nova_api_cron.service
    $ sudo systemctl stop tripleo_nova_compute.service
    $ sudo systemctl stop tripleo_nova_conductor.service
    $ sudo systemctl stop tripleo_nova_metadata.service
    $ sudo systemctl stop tripleo_nova_placement.service
    $ sudo systemctl stop tripleo_nova_scheduler.service
    ```

12. Optional: If you are using the Bare Metal Service (ironic) as the virt driver, you must manually update the service entries in your cell database for any bare metal instances whose **instances.host** is set to the controller that you are removing. Contact Red Hat Support for assistance.

NOTE

This manual update of the cell database when using Bare Metal Service (ironic) as the virt driver is a temporary workaround to ensure the nodes are rebalanced, until BZ2017980 is complete.

20.2. REMOVING A CEPH MONITOR DAEMON

If your Controller node is running a Ceph monitor service, complete the following steps to remove the **ceph-mon** daemon.

NOTE

Adding a new Controller node to the cluster also adds a new Ceph monitor daemon automatically.

Procedure

1. Connect to the Controller node that you want to replace:

```
$ ssh tripleo-admin@192.168.0.47
```

2. List the Ceph mon services:

```
$ sudo systemctl --type=service | grep ceph
ceph-4cf401f9-dd4c-5cda-9f0a-fa47fbf12b31@crash.controller-0.service          loaded active
running Ceph crash.controller-0 for 4cf401f9-dd4c-5cda-9f0a-fa47fbf12b31
  ceph-4cf401f9-dd4c-5cda-9f0a-fa47fbf12b31@mgr.controller-0.mufglq.service     loaded
active running Ceph mgr.controller-0.mufglq for 4cf401f9-dd4c-5cda-9f0a-fa47fbf12b31
  ceph-4cf401f9-dd4c-5cda-9f0a-fa47fbf12b31@mon.controller-0.service            loaded active
running Ceph mon.controller-0 for 4cf401f9-dd4c-5cda-9f0a-fa47fbf12b31
  ceph-4cf401f9-dd4c-5cda-9f0a-fa47fbf12b31@rgw.rgw.controller-0.ikaevh.service loaded
active running Ceph rgw.rgw.controller-0.ikaevh for 4cf401f9-dd4c-5cda-9f0a-fa47fbf12b31
```

3. Stop the Ceph mon service:

```
$ sudo systemtctl stop ceph-4cf401f9-dd4c-5cda-9f0a-fa47fbf12b31@mon.controller-
0.service
```

4. Disable the Ceph mon service:

```
$ sudo systemctl disable ceph-4cf401f9-dd4c-5cda-9f0a-fa47fbf12b31@mon.controller-
0.service
```

5. Disconnect from the Controller node that you want to replace.

6. Use SSH to connect to another Controller node in the same cluster:

```
$ ssh tripleo-admin@192.168.0.46
```

7. The Ceph specification file is modified and applied later in this procedure, to manipulate the file you must export it:

```
$ sudo cephadm shell --ceph orch ls --export > spec.yaml
```

8. Remove the monitor from the cluster:

```
$ sudo cephadm shell -- ceph mon remove controller-0
  removing mon.controller-0 at [v2:172.23.3.153:3300/0,v1:172.23.3.153:6789/0], there will be
2 monitors
```

9. Disconnect from the Controller node and log back into the Controller node you are removing from the cluster:

```
$ ssh tripleo-admin@192.168.0.47
```

10. List the Ceph mgr services:

```
$ sudo systemctl --type=service | grep ceph
ceph-4cf401f9-dd4c-5cda-9f0a-fa47fbf12b31@crash.controller-0.service          loaded active
running Ceph crash.controller-0 for 4cf401f9-dd4c-5cda-9f0a-fa47fbf12b31
  ceph-4cf401f9-dd4c-5cda-9f0a-fa47fbf12b31@mgr.controller-0.mufglq.service     loaded
active running Ceph mgr.controller-0.mufglq for 4cf401f9-dd4c-5cda-9f0a-fa47fbf12b31
  ceph-4cf401f9-dd4c-5cda-9f0a-fa47fbf12b31@rgw.rgw.controller-0.ikaevh.service loaded
active running Ceph rgw.rgw.controller-0.ikaevh for 4cf401f9-dd4c-5cda-9f0a-fa47fbf12b31
```

11. Stop the Ceph mgr service:

```
$ sudo systemctl stop ceph-4cf401f9-dd4c-5cda-9f0a-fa47fbf12b31@mgr.controller-
0.mufglq.service
```

12. Disable the Ceph mgr service:

```
$ sudo systemctl disable ceph-4cf401f9-dd4c-5cda-9f0a-fa47fbf12b31@mgr.controller-
0.mufglq.service
```

13. Start a **cephadm** shell:

```
$ sudo cephadm shell
```

14. Verify that the Ceph mgr service for the Controller node is removed from the cluster:

```
$ ceph -s
cluster:
     id:     b9b53581-d590-41ac-8463-2f50aa985001
     health: HEALTH_OK

   services:
     mon: 2 daemons, quorum controller-2,controller-1 (age 2h)
     mgr: controller-2(active, since 20h), standbys: controller1-1
     osd: 15 osds: 15 up (since 3h), 15 in (since 3h)

   data:
     pools:   3 pools, 384 pgs
```

```
    objects: 32 objects, 88 MiB
    usage:   16 GiB used, 734 GiB / 750 GiB avail
   pgs:     384 active+clean
```

The node is not listed if the Ceph mgr service is successfully removed.

15. Export the Red Hat Ceph Storage specification:

```
$ ceph orch ls --export > spec.yaml
```

16. In the **spec.yaml** specification file, remove all instances of the host, for example **controller-0**, from the **service_type: mon** and **service_type: mgr**.

17. Reapply the Red Hat Ceph Storage specification:

```
$ ceph orch apply -i spec.yaml
```

18. Verify that no Ceph daemons remain on the removed host:

```
$ ceph orch ps controller-0
```

NOTE

If daemons are present, use the following command to remove them:

```
$ ceph orch host drain controller-0
```

Prior to running the **ceph orch host drain** command, backup the contents of **/etc/ceph**. Restore the contents after running the **ceph orch host drain** command. You must back up prior to running the **ceph orch host drain** command until https://bugzilla.redhat.com/show_bug.cgi?id=2153827 is resolved.

19. Remove the **controller-0** host from the Red Hat Ceph Storage cluster:

```
$ ceph orch host rm controller-0
  Removed host 'controller-0'
```

20. Exit the cephadm shell:

```
$ exit
```

Additional Resources

For more information on controlling Red Hat Ceph Storage services with systemd, see Understanding process management for Ceph

For more information on editing and applying Red Hat Ceph Storage specification files, see Deploying the Ceph monitor daemons using the service specification

20.3. PREPARING THE CLUSTER FOR CONTROLLER NODE REPLACEMENT

Before you replace the node, ensure that Pacemaker is not running on the node and then remove that node from the Pacemaker cluster.

Procedure

1. To view the list of IP addresses for the Controller nodes, run the following command:

```
(undercloud)$ metalsmith -c Hostname -c "IP Addresses" list
+------------------------+-----------------------+
| Hostname               | IP Addresses          |
+------------------------+-----------------------+
| overcloud-compute-0    | ctlplane=192.168.0.44 |
| overcloud-controller-0 | ctlplane=192.168.0.47 |
| overcloud-controller-1 | ctlplane=192.168.0.45 |
| overcloud-controller-2 | ctlplane=192.168.0.46 |
+------------------------+-----------------------+
```

2. Log in to the node and confirm the pacemaker status. If pacemaker is running, use the **pcs cluster** command to stop pacemaker. This example stops pacemaker on **overcloud-controller-0**:

```
(undercloud) $ ssh tripleo-admin@192.168.0.45 "sudo pcs status | grep -w Online | grep -w overcloud-controller-0"
(undercloud) $ ssh tripleo-admin@192.168.0.45 "sudo pcs cluster stop overcloud-controller-0"
```

NOTE

In the case that the node is physically unavailable or stopped, it is not necessary to perform the previous operation, as pacemaker is already stopped on that node.

3. After you stop Pacemaker on the node, delete the node from the pacemaker cluster. The following example logs in to **overcloud-controller-1** to remove **overcloud-controller-0**:

```
(undercloud) $ ssh tripleo-admin@192.168.0.45 "sudo pcs cluster node remove overcloud-controller-0"
```

If the node that that you want to replace is unreachable (for example, due to a hardware failure), run the **pcs** command with additional **--skip-offline** and **--force** options to forcibly remove the node from the cluster:

```
(undercloud) $ ssh tripleo-admin@192.168.0.45 "sudo pcs cluster node remove overcloud-controller-0 --skip-offline --force"
```

4. After you remove the node from the pacemaker cluster, remove the node from the list of known hosts in pacemaker:

```
(undercloud) $ ssh tripleo-admin@192.168.0.45 "sudo pcs host deauth overcloud-controller-0"
```

You can run this command whether the node is reachable or not.

5. To ensure that the new Controller node uses the correct STONITH fencing device after replacement, delete the devices from the node by entering the following command:

   ```
   (undercloud) $ ssh tripleo-admin@192.168.0.45 "sudo pcs stonith delete
   <stonith_resource_name>"
   ```

   - Replace **<stonith_resource_name>** with the name of the STONITH resource that corresponds to the node. The resource name uses the the format **<resource_agent>-<host_mac>**. You can find the resource agent and the host MAC address in the **FencingConfig** section of the **fencing.yaml** file.

6. The overcloud database must continue to run during the replacement procedure. To ensure that Pacemaker does not stop Galera during this procedure, select a running Controller node and run the following command on the undercloud with the IP address of the Controller node:

   ```
   (undercloud) $ ssh tripleo-admin@192.168.0.45 "sudo pcs resource unmanage galera-
   bundle"
   ```

7. Remove the OVN northbound database server for the replaced Controller node from the cluster:

   a. Obtain the server ID of the OVN northbound database server to be replaced:

      ```
      $ ssh tripleo-admin@<controller_ip> sudo podman exec ovn_cluster_north_db_server
      ovs-appctl -t /var/run/ovn/ovnnb_db.ctl cluster/status OVN_Northbound 2>/dev/null|grep -
      A4 Servers:
      ```

      Replace **<controller_ip>** with the IP address of any active Controller node.

      You should see output similar to the following:

      ```
      Servers:
      96da (96da at tcp:172.17.1.55:6643) (self) next_index=26063 match_index=26063 466b
      (466b at tcp:172.17.1.51:6643) next_index=26064 match_index=26063 last msg 2936
      ms ago
      ba77 (ba77 at tcp:172.17.1.52:6643) next_index=26064 match_index=26063 last msg
      2936 ms ago
      ```

      In this example, **172.17.1.55** is the internal IP address of the Controller node that is being replaced, so the northbound database server ID is **96da**.

   b. Using the server ID you obtained in the preceding step, remove the OVN northbound database server by running the following command:

      ```
      $ ssh tripleo-admin@172.17.1.52 sudo podman exec ovn_cluster_north_db_server ovs-
      appctl -t /var/run/ovn/ovnnb_db.ctl cluster/kick OVN_Northbound 96da
      ```

      In this example, you would replace **172.17.1.52** with the IP address of any active Controller node, and replace **96da** with the server ID of the OVN northbound database server.

8. Remove the OVN southbound database server for the replaced Controller node from the cluster:

   a. Obtain the server ID of the OVN southbound database server to be replaced:

```
$ ssh tripleo-admin@<controller_ip> sudo podman exec ovn_cluster_north_db_server
ovs-appctl -t /var/run/ovn/ovnnb_db.ctl cluster/status OVN_Southbound 2>/dev/null|grep
-A4 Servers:
```

Replace **<controller_ip>** with the IP address of any active Controller node.

You should see output similar to the following:

```
Servers:
e544 (e544 at tcp:172.17.1.55:6644) last msg 42802690 ms ago
17ca (17ca at tcp:172.17.1.51:6644) last msg 5281 ms ago
6e52 (6e52 at tcp:172.17.1.52:6644) (self)
```

In this example, **172.17.1.55** is the internal IP address of the Controller node that is being replaced, so the southbound database server ID is **e544**.

b. Using the server ID you obtained in the preceding step, remove the OVN southbound database server by running the following command:

```
$ ssh tripleo-admin@172.17.1.52 sudo podman exec ovn_cluster_south_db_server ovs-
appctl -t /var/run/ovn/ovnsb_db.ctl cluster/kick OVN_Southbound e544
```

In this example, you would replace **172.17.1.52** with the IP address of any active Controller node, and replace **e544** with the server ID of the OVN southbound database server.

9. Run the following clean up commands to prevent cluster rejoins.
Substitute **<replaced_controller_ip>** with the IP address of the Controller node that you are replacing:

```
$ ssh tripleo-admin@<replaced_controller_ip> sudo systemctl disable --now
tripleo_ovn_cluster_south_db_server.service tripleo_ovn_cluster_north_db_server.service

$ ssh tripleo-admin@<replaced_controller_ip> sudo rm -rfv /var/lib/openvswitch/ovn/.ovn*
/var/lib/openvswitch/ovn/ovn*.db
```

20.4. REPLACING A BOOTSTRAP CONTROLLER NODE

If you want to replace the Controller node that you use for bootstrap operations and keep the node name, complete the following steps to set the name of the bootstrap Controller node after the replacement process.

IMPORTANT

Currently, when a bootstrap Controller node is replaced, the OVN database cluster is partitioned with two database clusters for both the northbound and southbound databases. This situation makes instances unusable.

To find the name of the bootstrap Controller node, run the following command:

```
ssh tripleo-admin@<controller_ip> "sudo hiera -c /etc/puppet/hiera.yaml
ovn_dbs_short_bootstrap_node_name"
```

Workaround: Do not reuse the original bootstrap node hostname and IP address for the new Controller node. RHOSP director sorts the hostnames and then selects the first hostname in the list as the bootstrap node. Choose a name for the new Controller node so that it does not become the first hostname after sorting.

You can track the progress of the fix for this known issue in BZ 2222543.

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.

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 .

20.5. UNPROVISION AND REMOVE CONTROLLER NODES

To unprovision and remove Controller nodes, complete the following steps.

Procedure

1. Source the **stackrc** file:

   ```
   $ source ~/stackrc
   ```

2. Identify the UUID of the **overcloud-controller-0** node:

```
(undercloud)$ NODE=$(metalsmith -c UUID -f value show overcloud-controller-0)
```

3. Set the node to maintenance mode:

```
$ openstack baremetal node maintenance set $NODE
```

4. Copy the **overcloud-baremetal-deploy.yaml** file:

```
$ cp /home/stack/templates/overcloud-baremetal-deploy.yaml
/home/stack/templates/unprovision_controller-0.yaml
```

5. In the **unprovision_controller-0.yaml** file, lower the Controller count to unprovision the Controller node that you are replacing. In this example, the count is reduced from **3** to **2**. Move the **controller-0** node to the **instances** dictionary and set the **provisioned** parameter to **false**:

```
- name: Controller
  count: 2
  hostname_format: controller-%index%
  defaults:
    resource_class: BAREMETAL.controller
    networks:
      [ ... ]
  instances:
  - hostname: controller-0
    name: <IRONIC_NODE_UUID_or_NAME>
    provisioned: false
- name: Compute
  count: 2
  hostname_format: compute-%index%
  defaults:
    resource_class: BAREMETAL.compute
    networks:
      [ ... ]
```

6. Run the **node unprovision** command:

```
$ openstack overcloud node delete \
  --stack overcloud \
  --baremetal-deployment /home/stack/templates/unprovision_controller-0.yaml
```

```
The following nodes will be unprovisioned:
+--------------+-------------------------+--------------------------------------+
| hostname     | name                    | id                                   |
+--------------+-------------------------+--------------------------------------+
| controller-0 | baremetal-35400-leaf1-2 | b0d5abf7-df28-4ae7-b5da-9491e84c21ac |
+--------------+-------------------------+--------------------------------------+

Are you sure you want to unprovision these overcloud nodes and ports [y/N]?
```

Optional

Delete the ironic node:

```
$ openstack baremetal node delete <IRONIC_NODE_UUID>
```

- Replace **IRONIC_NODE_UUID** with the UUID of the node.

20.6. DEPLOYING A NEW CONTROLLER NODE TO THE OVERCLOUD

To deploy a new controller node to the overcloud complete the following steps.

Prerequisites

- The new Controller node must be registered, inspected, and tagged ready for provisioning. For more information, see Provisioning bare metal overcloud nodes

Procedure

1. Log into director and source the **stackrc** credentials file:

   ```
   $ source ~/stackrc
   ```

2. Provision the overcloud with the original **overcloud-baremetal-deploy.yaml** environment file:

   ```
   $ openstack overcloud node provision
     --stack overcloud
     --network-config
     --output /home/stack/templates/overcloud-baremetal-deployed.yaml
     /home/stack/templates/overcloud-baremetal-deploy.yaml
   ```

NOTE

If you want to use the same scheduling, placement, or IP addresses you can edit the **overcloud-baremetal-deploy.yaml** environment file. Set the hostname, name, and networks for the new controller-0 instance in the **instances** section. For example:

```
- name: Controller
  count: 3
  hostname_format: controller-%index%
  defaults:
    resource_class: BAREMETAL.controller
    networks:
    - network: external
      subnet: external_subnet
    - network: internal_api
      subnet: internal_api_subnet01
    - network: storage
      subnet: storage_subnet01
    - network: storage_mgmt
      subnet: storage_mgmt_subnet01
    - network: tenant
      subnet: tenant_subnet01
    network_config:
      template: templates/multiple_nics/multiple_nics_dvr.j2
      default_route_network:
      - external
  instances:
  - hostname: controller-0
    name: baremetal-35400-leaf1-2
    networks:
    - network: external
      subnet: external_subnet
      fixed_ip: 10.0.0.224
    - network: internal_api
      subnet: internal_api_subnet01
      fixed_ip: 172.17.0.97
    - network: storage
      subnet: storage_subnet01
      fixed_ip: 172.18.0.24
    - network: storage_mgmt
      subnet: storage_mgmt_subnet01
      fixed_ip: 172.19.0.129
    - network: tenant
      subnet: tenant_subnet01
      fixed_ip: 172.16.0.11
- name: Compute
  count: 2
  hostname_format: compute-%index%
  defaults:
    [ ... ]
```

When the node is provisioned, remove the **instances** section from the **overcloud-baremetal-deploy.yaml** file.

3. To create the **cephadm** user on the new Controller node, export a basic Ceph specification containing the new host information:

```
$ openstack overcloud ceph spec --stack overcloud \
  /home/stack/templates/overcloud-baremetal-deployed.yaml \
  -o ceph_spec_host.yaml
```

NOTE

If your environment uses a custom role, include the **--roles-data** option.

4. Add the **cephadm** user to the new Controller node:

```
$ openstack overcloud ceph user enable \
  --stack overcloud ceph_spec_host.yaml
```

5. Add the new role to the Ceph cluster:

```
$ sudo cephadm shell \
  -- ceph orch test add controlller-3 <IP_ADDRESS> <LABELS>
192.168.24.31 _admin mon mgr
Inferring fsid 4cf401f9-dd4c-5cda-9f0a-fa47fbf12b31
Using recent ceph image undercloud-0.ctlplane.redhat.local:8787/rh-
osbs/rhceph@sha256:3075e8708792ebd527ca14849b6af4a11256a3f881ab09b837d7af0f8b2
102ea
Added host 'controller-3' with addr '192.168.24.31'
```

- Replace <IP_ADDRESS> with the IP address of the Controller node.
- Replace <LABELS> with any required Ceph labels.

6. Re-run the **openstack overcloud deploy** command:

```
$ openstack overcloud deploy --stack overcloud --templates \
    -n /home/stack/templates/network_data.yaml \
    -r /home/stack/templates/roles_data.yaml \
    -e /home/stack/templates/overcloud-baremetal-deployed.yaml \
    -e /home/stack/templates/overcloud-networks-deployed.yaml \
    -e /home/stack/templates/overcloud-vips-deployed.yaml \
    -e /home/stack/templates/bootstrap_node.yaml \
    -e [ ... ]
```

NOTE

If the replacement Controller node is the bootstrap node, include the **bootstrap_node.yaml** environment file.

20.7. DEPLOYING CEPH SERVICES ON THE NEW CONTROLLER NODE

After you provision a new Controller node and the Ceph monitor services are running you can deploy the **mgr**, **rgw** and **osd** Ceph services on the Controller node.

Prerequisites

- The new Controller node is provisioned and is running Ceph monitor services.

Procedure

1. Modify the **spec.yml** environment file, replace the previous Controller node name with the new Controller node name:

```
$ cephadm shell -- ceph orch ls --export > spec.yml
```

NOTE

Do not use the basic Ceph environment file **ceph_spec_host.yaml** as it does not contain all necessary cluster information.

2. Apply the modified Ceph specification file:

```
$ cat spec.yml | sudo cephadm shell -- ceph orch apply -i -
Inferring fsid 4cf401f9-dd4c-5cda-9f0a-fa47fbf12b31
Using recent ceph image undercloud-0.ctlplane.redhat.local:8787/rh-
osbs/rhceph@sha256:3075e8708792ebd527ca14849b6af4a11256a3f881ab09b837d7af0f8b2
102ea
Scheduled crash update...
Scheduled mgr update...
Scheduled mon update...
Scheduled osd.default_drive_group update...
Scheduled rgw.rgw update...
```

3. Verify the visibility of the new monitor:

```
$ sudo cephadm --ceph status
```

.

20.8. CLEANING UP AFTER CONTROLLER NODE REPLACEMENT

After you complete the node replacement, you can finalize the Controller cluster.

Procedure

1. Log into a Controller node.

2. Enable Pacemaker management of the Galera cluster and start Galera on the new node:

```
[tripleo-admin@overcloud-controller-0 ~]$ sudo pcs resource refresh galera-bundle
[tripleo-admin@overcloud-controller-0 ~]$ sudo pcs resource manage galera-bundle
```

3. Enable fencing:

```
[tripleo-admin@overcloud-controller-0 ~]$ sudo pcs property set stonith-enabled=true
```

4. Perform a final status check to ensure that the services are running correctly:

```
[tripleo-admin@overcloud-controller-0 ~]$ sudo pcs status
```

> **NOTE**
>
> If any services have failed, use the **pcs resource refresh** command to resolve and restart the failed services.

5. Exit to director:

   ```
   [tripleo-admin@overcloud-controller-0 ~]$ exit
   ```

6. Source the **overcloudrc** file so that you can interact with the overcloud:

   ```
   $ source ~/overcloudrc
   ```

7. Check the network agents in your overcloud environment:

   ```
   (overcloud) $ openstack network agent list
   ```

8. If any agents appear for the old node, remove them:

   ```
   (overcloud) $ for AGENT in $(openstack network agent list --host overcloud-controller-
   1.localdomain -c ID -f value) ; do openstack network agent delete $AGENT ; done
   ```

9. If necessary, add your router to the L3 agent host on the new node. Use the following example command to add a router named **r1** to the L3 agent using the UUID 2d1c1dc1-d9d4-4fa9-b2c8-f29cd1a649d4:

   ```
   (overcloud) $ openstack network agent add router --l3 2d1c1dc1-d9d4-4fa9-b2c8-
   f29cd1a649d4 r1
   ```

10. Clean the cinder services.

    a. List the cinder services:

       ```
       (overcloud) $ openstack volume service list
       ```

    b. Log in to a controller node, connect to the **cinder-api** container and use the **cinder-manage service remove** command to remove leftover services:

       ```
       [tripleo-admin@overcloud-controller-0 ~]$ sudo podman exec -it cinder_api cinder-
       manage service remove cinder-backup <host>
       [tripleo-admin@overcloud-controller-0 ~]$ sudo podman exec -it cinder_api cinder-
       manage service remove cinder-scheduler <host>
       ```

11. Clean the RabbitMQ cluster.

    a. Log into a Controller node.

    b. Use the **podman exec** command to launch bash, and verify the status of the RabbitMQ cluster:

```
[tripleo-admin@overcloud-controller-0 ~]$ podman exec -it rabbitmq-bundle-podman-0
bash
[tripleo-admin@overcloud-controller-0 ~]$ rabbitmqctl cluster_status
```

c. Use the **rabbitmqctl** command to forget the replaced controller node:

```
[tripleo-admin@overcloud-controller-0 ~]$ rabbitmqctl forget_cluster_node
<node_name>
```

12. If you replaced a bootstrap Controller node, you must remove the environment file **~/templates/bootstrap-controller.yaml** after the replacement process, or delete the **pacemaker_short_bootstrap_node_name** and **mysql_short_bootstrap_node_name** parameters from your existing environment file. This step prevents director from attempting to override the Controller node name in subsequent replacements. For more information, see Replacing a bootstrap Controller node .

13. If you are using the Object Storage service (swift) on the overcloud, you must synchronize the swift rings after updating the overcloud nodes. Use a script, similar to the following example, to distribute ring files from a previously existing Controller node (Controller node 0 in this example) to all Controller nodes and restart the Object Storage service containers on those nodes:

```
#!/bin/sh
set -xe

SRC="tripleo-admin@overcloud-controller-0.ctlplane"
ALL="tripleo-admin@overcloud-controller-0.ctlplane tripleo-admin@overcloud-controller-
1.ctlplane tripleo-admin@overcloud-controller-2.ctlplane"
```

- Fetch the current set of ring files:

```
ssh "${SRC}" 'sudo tar -czvf - /var/lib/config-data/puppet-
generated/swift_ringbuilder/etc/swift/{*.builder,*.ring.gz,backups/*.builder}' > swift-
rings.tar.gz
```

- Upload rings to all nodes, put them into the correct place, and restart swift services:

```
for DST in ${ALL}; do
  cat swift-rings.tar.gz | ssh "${DST}" 'sudo tar -C / -xvzf -'
  ssh "${DST}" 'sudo podman restart swift_copy_rings'
  ssh "${DST}" 'sudo systemctl restart tripleo_swift*'
done
```

CHAPTER 21. 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.

21.1. 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.

21.2. REBOOTING CONTROLLER AND COMPOSABLE NODES

Reboot Controller nodes and standalone nodes based on composable roles, and exclude Compute nodes and Ceph Storage nodes.

Procedure

1. Log in to the node that you want to reboot.
2. Optional: If the node uses Pacemaker resources, stop the cluster:

   ```
   [tripleo-admin@overcloud-controller-0 ~]$ sudo pcs cluster stop
   ```

3. Reboot the node:

```
[tripleo-admin@overcloud-controller-0 ~]$ sudo reboot
```

4. Wait until the node boots.

Verification

1. Verify that the services are enabled.
   a. If the node uses Pacemaker services, check that the node has rejoined the cluster:

      ```
      [tripleo-admin@overcloud-controller-0 ~]$ sudo pcs status
      ```

   b. If the node uses Systemd services, check that all services are enabled:

      ```
      [tripleo-admin@overcloud-controller-0 ~]$ sudo systemctl status
      ```

   c. If the node uses containerized services, check that all containers on the node are active:

      ```
      [tripleo-admin@overcloud-controller-0 ~]$ sudo podman ps
      ```

21.3. 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.

21.4. REBOOTING A CEPH STORAGE (OSD) CLUSTER

Complete the following steps to reboot a cluster of Ceph Storage (OSD) nodes.

Prerequisites

- On a Ceph Monitor or Controller node that is running the **ceph-mon** service, check that the Red Hat Ceph Storage cluster status is healthy and the pg status is **active+clean**:

  ```
  $ sudo cephadm -- shell ceph status
  ```

  If the Ceph cluster is healthy, it returns a status of **HEALTH_OK**.

  If the Ceph cluster status is unhealthy, it returns a status of **HEALTH_WARN** or **HEALTH_ERR**. For troubleshooting guidance, see the Red Hat Ceph Storage 5 Troubleshooting Guide .

Procedure

1. Log in to a Ceph Monitor or Controller node that is running the **ceph-mon** service, and disable Ceph Storage cluster rebalancing temporarily:

   ```
   $ sudo cephadm shell -- ceph osd set noout
   $ sudo cephadm shell -- ceph osd set norebalance
   ```

   NOTE

   If you have a multistack or distributed compute node (DCN) architecture, you must specify the Ceph cluster name when you set the **noout** and **norebalance** flags. For example: **sudo cephadm shell -c /etc/ceph/<cluster>.conf -k /etc/ceph/<cluster>.client.keyring**.

2. Select the first Ceph Storage node that you want to reboot and log in to the node.

3. Reboot the node:

   ```
   $ sudo reboot
   ```

4. Wait until the node boots.

5. Log in to the node and check the Ceph cluster status:

   ```
   $ sudo cephadm -- shell ceph status
   ```

   Check that the **pgmap** reports all **pgs** as normal (**active+clean**).

6. Log out of the node, reboot the next node, and check its status. Repeat this process until you have rebooted all Ceph Storage nodes.

7. When complete, log in to a Ceph Monitor or Controller node that is running the **ceph-mon** service and enable Ceph cluster rebalancing:

   ```
   $ sudo cephadm shell -- ceph osd unset noout
   $ sudo cephadm shell -- ceph osd unset norebalance
   ```

   NOTE

   If you have a multistack or distributed compute node (DCN) architecture, you must specify the Ceph cluster name when you unset the **noout** and **norebalance** flags. For example: **sudo cephadm shell -c /etc/ceph/<cluster>.conf -k /etc/ceph/<cluster>.client.keyring**

8. Perform a final status check to verify that the cluster reports **HEALTH_OK**:

   ```
   $ sudo cephadm shell ceph status
   ```

21.5. 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.

21.6. REBOOTING COMPUTE NODES

To ensure minimal downtime of instances in your Red Hat OpenStack Platform environment, the Migrating instances workflow outlines the steps you must complete to migrate instances from the Compute node that you want to reboot.

Migrating instances workflow

1. Decide whether to migrate instances to another Compute node before rebooting the node.
2. Select and disable the Compute node that you want to reboot so that it does not provision new instances.
3. Migrate the instances to another Compute node.
4. Reboot the empty Compute node.
5. Enable the empty Compute node.

Prerequisites

- Before you reboot the Compute node, you must decide whether to migrate instances to another Compute node while the node is rebooting.
  Review the list of migration constraints that you might encounter when you migrate virtual machine instances between Compute nodes. For more information, see Migration constraints in *Configuring the Compute Service for Instance Creation* .

- If you cannot migrate the instances, you can set the following core template parameters to control the state of the instances after the Compute node reboots:

  NovaResumeGuestsStateOnHostBoot

  Determines whether to return instances to the same state on the Compute node after reboot. When set to **False**, the instances remain down and you must start them manually. The default value is **False**.

  NovaResumeGuestsShutdownTimeout

  Number of seconds to wait for an instance to shut down before rebooting. It is not recommended to set this value to **0**. The default value is **300**.
  For more information about overcloud parameters and their usage, see Overcloud Parameters.

Procedure

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

2. List all Compute nodes and their UUIDs:

   ```
   $ source ~/stackrc
   (undercloud) $ metalsmith list | grep compute
   ```

   Identify the UUID of the Compute node that you want to reboot.

3. From the overcloud, select a Compute node and disable it:

   ```
   $ source ~/overcloudrc
   (overcloud)$ openstack compute service list
   (overcloud)$ openstack compute service set <hostname> nova-compute --disable
   ```

   - Replace **<hostname>** with the hostname of your Compute node.

4. List all instances on the Compute node:

   ```
   (overcloud)$ openstack server list --host <hostname> --all-projects
   ```

5. Optional: To migrate the instances to another Compute node, complete the following steps:

   a. If you decide to migrate the instances to another Compute node, use one of the following commands:

      - To migrate the instance to a different host, run the following command:

        ```
        (overcloud) $ openstack server migrate <instance_id> --live <target_host> --wait
        ```

        - Replace **<instance_id>** with your instance ID.
        - Replace **<target_host>** with the host that you are migrating the instance to.

      - Let **nova-scheduler** automatically select the target host:

        ```
        (overcloud) $ nova live-migration <instance_id>
        ```

      - Live migrate all instances at once:

        ```
        $ nova host-evacuate-live <hostname>
        ```

        NOTE

        The **nova** command might cause some deprecation warnings, which are safe to ignore.

   b. Wait until migration completes.

   c. Confirm that the migration was successful:

      ```
      (overcloud) $ openstack server list --host <hostname> --all-projects
      ```

   d. Continue to migrate instances until none remain on the Compute node.

6. Log in to the Compute node and reboot the node:

```
[tripleo-admin@overcloud-compute-0 ~]$ sudo reboot
```

7. Wait until the node boots.

8. Re-enable the Compute node:

```
$ source ~/overcloudrc
(overcloud) $ openstack compute service set <hostname>  nova-compute --enable
```

9. Check that the Compute node is enabled:

```
(overcloud) $ openstack compute service list
```

CHAPTER 22. 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

22.1. 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

22.2. 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.

22.3. 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.

22.4. 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
   ```

22.5. 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?"

22.6. 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.

22.7. 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
```

22.8. 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.

22.9. 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.

22.10. STARTING THE UNDERCLOUD

As a part of starting the Red Hat OpenStack Platform environment, power on the undercloud node, log in to the undercloud, and check the undercloud services.

Prerequisites

- The undercloud is powered down.

Procedure

- Power on the undercloud and wait until the undercloud boots.

Verification

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

   ```
   $ source ~/stackrc
   ```

3. Check the services on the undercloud:

   ```
   $ systemctl list-units 'tripleo_*'
   ```

4. Validate the static inventory file named **tripleo-ansible-inventory.yaml**:

   ```
   $ validation run --group pre-introspection -i <inventory_file>
   ```

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

NOTE

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

5. Check that all services and containers are active and healthy:

```
$ validation run --validation service-status --limit undercloud -i <inventory_file>
```

Additional resources

- Using the validation framework

22.11. STARTING CONTROLLER NODES

As a part of starting the Red Hat OpenStack Platform environment, power on each Controller node and check the non-Pacemaker services on the node.

Prerequisites

- The Controller nodes are powered down.

Procedure

- Power on each Controller node.

Verification

1. Log in to each Controller node as the **root** user.
2. Check the services on the Controller node:

```
$ systemctl -t service
```

   Only non-Pacemaker based services are running.

3. Wait until the Pacemaker services start and check that the services started:

```
$ pcs status
```

NOTE

If your environment uses Instance HA, the Pacemaker resources do not start until you start the Compute nodes or perform a manual unfence operation with the **pcs stonith confirm <compute_node>** command. You must run this command on each Compute node that uses Instance HA.

22.12. 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?"

22.13. 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
```

22.14. 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 23. 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.

23.1. 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.

23.2. 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.

23.3. 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
|
+--------------------------------------+------------------------------------------------------------------------
------------------------------------------------+
```

23.4. 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 Director configuration parameters.

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 24. 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**.

24.1. 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
   ```

24.2. 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
```

24.3. 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 25. CONFIGURING AUTOMATIC PROFILE TAGGING

The introspection process performs a series of benchmark tests. 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 under-performing or unstable nodes and isolate these nodes from use in the overcloud.
- The policies can define whether to tag nodes into specific profiles automatically.

25.1. 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"
  }
]
```

25.2. 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.

25.3. IMPORTING POLICY FILES INTO DIRECTOR

To apply the policy rules you defined in your policy JSON file, you must import the policy file into director.

Procedure

1. Import the policy file into director:

```
$ openstack baremetal introspection rule import <policy_file>
```

- Replace **<policy_file>** with the name of your policy rule file, for example, **rules.json**.

2. Run the introspection process:

```
$ openstack overcloud node introspect --all-manageable
```

3. Retrieve the UUIDs of the nodes that the policy rules are applied to:

```
$ openstack baremetal node list
```

4. Confirm that the nodes have been assigned the profiles defined in your policy rule file:

```
$ openstack baremetal node show <node_uuid>
```

5. If you made a mistake in introspection rules, then delete all rules:

```
$ openstack baremetal introspection rule purge
```

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

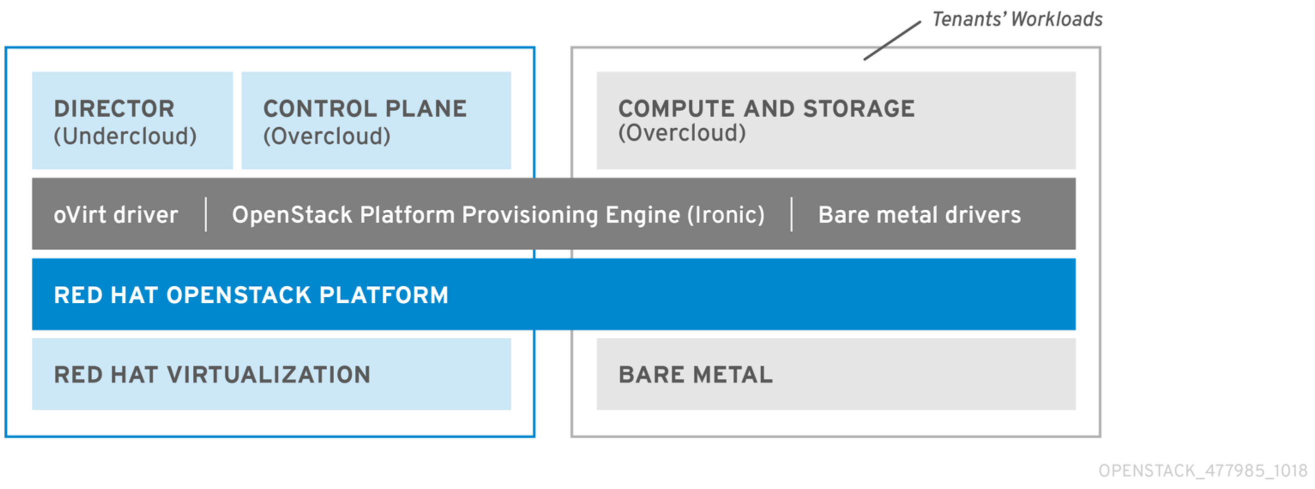
Virtualized Controller nodes are supported only on Red Hat Virtualization.

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


Tenants' Workloads
DIRECTOR (Undercloud)
CONTROL PLANE (Overcloud)
COMPUTE AND STORAGE (Overcloud)
oVirt driver
OpenStack Platform Provisioning Engine (Ironic)
Bare metal drivers
RED HAT OPENSTACK PLATFORM
RED HAT VIRTUALIZATION
BARE METAL
OPENSTACK_477985_1018

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 .
- 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-rhel9/openstack-keystone:17.0
```

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-rhel9
        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-rhel9/openstack-
keystone:17.0
```

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

The Bare Metal Provisioning inspector service, **ironic-inspector**, times out after a default one-hour period if the inspection RAM disk does not respond. The timeout might indicate a bug in the inspection RAM disk, but usually the timeout occurs due to an environment misconfiguration.

You can diagnose and resolve common environment misconfiguration issues to ensure the introspection process runs to completion.

Procedure

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

```
$ source ~/stackrc
```

2. Ensure that your nodes are in a **manageable** state. The introspection does not inspect nodes in an **available** state, which is meant for deployment. If you want to inspect nodes that are in an **available** state, change the node status to **manageable** state before introspection:

```
(undercloud)$ openstack baremetal node manage <node_uuid>
```

3. To configure temporary access to the introspection RAM disk during introspection debugging, use the **sshkey** parameter to append your public SSH key to the **kernel** configuration in the **/httpboot/inspector.ipxe** file:

```
kernel http://192.2.0.1:8088/agent.kernel ipa-inspection-callback-
url=http://192.168.0.1:5050/v1/continue ipa-inspection-collectors=default,extra-hardware,logs
systemd.journald.forward_to_console=yes BOOTIF=${mac} ipa-debug=1 ipa-inspection-
benchmarks=cpu,mem,disk selinux=0 sshkey="<public_ssh_key>"
```

4. Run the introspection on the node:

```
(undercloud)$ openstack overcloud node introspect <node_uuid> --provide
```

   Use the **--provide** option to change the node state to **available** after the introspection completes.

5. Identify the IP address of the node from the **dnsmasq** logs:

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

6. If an error occurs, access the node using the root user and temporary access details:

```
$ ssh root@192.168.24.105
```

   Access the node during introspection to run diagnostic commands and troubleshoot the introspection failure.

7. To stop the introspection process, run the following command:

```
(undercloud)$ openstack baremetal introspection abort <node_uuid>
```

   You can also wait until the process times out.

NOTE

Red Hat OpenStack Platform director retries introspection three times after the initial abort. Run the **openstack baremetal introspection abort** command at each attempt to abort the introspection completely.

28.3. TROUBLESHOOTING OVERCLOUD CREATION AND DEPLOYMENT

The initial creation of the overcloud occurs with the OpenStack Orchestration (heat) service. If an overcloud deployment fails, use the OpenStack clients and service log files to diagnose the failed deployment.

Procedure

1. Source the **stackrc** file:

   ```
   $ source ~/stackrc
   ```

2. 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
   ```

28.4. TROUBLESHOOTING NODE PROVISIONING

The OpenStack Orchestration (heat) service controls the provisioning process. If node provisioning fails, use the OpenStack clients and service log files to diagnose the issues.

Procedure

1. Source the **stackrc** file:

   ```
   $ source ~/stackrc
   ```

2. Check the bare metal service to see all registered nodes and their current status:

   ```
   (undercloud) $ openstack baremetal node list

   +----------+------+---------------+-------------+-----------------+-------------+
   | UUID     | Name | Instance UUID | Power State | Provision State | Maintenance |
   +----------+------+---------------+-------------+-----------------+-------------+
   | f1e261...| None | None          | power off   | available       | False       |
   | f0b8c1...| None | None          | power off   | available       | False       |
   +----------+------+---------------+-------------+-----------------+-------------+
   ```

   All nodes available for provisioning should have the following states set:

   - **Maintenance** set to **False**.
   - **Provision State** set to **available** before provisioning.

3. If a node does not have **Maintenance** set to **False** or **Provision State** set to **available**, then use the following table to identify the problem and the solution:

Problem	Cause	Solution
Maintenance sets itself to **True** automatically.	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

28.5. TROUBLESHOOTING IP ADDRESS CONFLICTS DURING PROVISIONING

Introspection and deployment tasks fail if the destination hosts are allocated an IP address that is already in use. To prevent these failures, you can perform a port scan of the Provisioning network to determine whether the discovery IP range and host IP range are free.

Procedure

1. Install **nmap**:

```
$ sudo dnf install nmap
```

2. Use **nmap** to scan the IP address range for active addresses. This example scans the 192.168.24.0/24 range, replace this with the IP subnet of the Provisioning network (using CIDR bitmask notation):

   ```
   $ sudo nmap -sn 192.168.24.0/24
   ```

3. Review the output of the **nmap** scan. For example, you should see the IP address of the undercloud, and any other hosts that are present on the subnet:

   ```
   $ sudo nmap -sn 192.168.24.0/24

   Starting Nmap 6.40 ( http://nmap.org ) at 2015-10-02 15:14 EDT
   Nmap scan report for 192.168.24.1
   Host is up (0.00057s latency).
   Nmap scan report for 192.168.24.2
   Host is up (0.00048s latency).
   Nmap scan report for 192.168.24.3
   Host is up (0.00045s latency).
   Nmap scan report for 192.168.24.5
   Host is up (0.00040s latency).
   Nmap scan report for 192.168.24.9
   Host is up (0.00019s latency).
   Nmap done: 256 IP addresses (5 hosts up) scanned in 2.45 seconds
   ```

   If any of the active IP addresses conflict with the IP ranges in undercloud.conf, you must either change the IP address ranges or release the IP addresses before you introspect or deploy the overcloud nodes.

28.6. 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
```

28.7. TROUBLESHOOTING CONTAINER CONFIGURATION

Red Hat OpenStack Platform director uses **podman** to manage containers and **puppet** to create container configuration. This procedure shows how to diagnose a container when errors occur.

Accessing the host

1. Source the **stackrc** file:

```
$ source ~/stackrc
```

2. Get the IP address of the node with the container failure.

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

3. Log in to the node:

```
(undercloud) $ ssh tripleo-admin@192.168.24.60
```

Identifying failed containers

1. View all containers:

```
$ sudo podman ps --all
```

Identify the failed container. The failed container usually exits with a non-zero status.

Checking container logs

1. Each container retains standard output from its main process. Use this output as a log to help determine what actually occurs during a container run. For example, to view the log for the **keystone** container, run the following command:

   ```
   $ sudo podman logs keystone
   ```

   In most cases, this log contains information about the cause of a container failure.

2. The host also retains the **stdout** log for the failed service. You can find the **stdout** logs in **/var/log/containers/stdouts/**. For example, to view the log for a failed **keystone** container, run the following command:

   ```
   $ cat /var/log/containers/stdouts/keystone.log
   ```

Inspecting containers

In some situations, you might need to verify information about a container. For example, use the following command to view **keystone** container data:

```
$ sudo podman inspect keystone
```

This command returns a JSON object containing low-level configuration data. You can pipe the output to the **jq** command to parse specific data. For example, to view the container mounts for the **keystone** container, run the following command:

```
$ sudo podman inspect keystone | jq .[0].Mounts
```

You can also use the **--format** option to parse data to a single line, which is useful for running commands against sets of container data. For example, to recreate the options used to run the **keystone** container, use the following **inspect** command with the **--format** option:

```
$ sudo podman inspect --format='{{range .Config.Env}} -e "{{.}}" {{end}} {{range .Mounts}} -v
{{.Source}}:{{.Destination}}:{{ join .Options "," }}{{end}} -ti {{.Config.Image}}' keystone
```

NOTE

The **--format** option uses Go syntax to create queries.

Use these options in conjunction with the **podman run** command to recreate the container for troubleshooting purposes:

```
$ OPTIONS=$( sudo podman inspect --format='{{range .Config.Env}} -e "{{.}}" {{end}} {{range
.Mounts}} -v {{.Source}}:{{.Destination}}{{if .Mode}}:{{.Mode}}{{end}}{{end}} -ti {{.Config.Image}}'
keystone )
$ sudo podman run --rm $OPTIONS /bin/bash
```

Running commands in a container

In some cases, you might need to obtain information from within a container through a specific Bash command. In this situation, use the following **podman** command to execute commands within a running container. For example, run the **podman exec** command to run a command inside the **keystone** container:

```
$ sudo podman exec -ti keystone <COMMAND>
```

NOTE

The **-ti** options run the command through an interactive pseudoterminal.

- Replace **<COMMAND>** with the command you want to run. For example, each container has a health check script to verify the service connection. You can run the health check script for **keystone** with the following command:

```
$ sudo podman exec -ti keystone /openstack/healthcheck
```

To access the container shell, run **podman exec** using **/bin/bash** as the command you want to run inside the container:

```
$ sudo podman exec -ti keystone /bin/bash
```

Viewing a container filesystem

1. To view the file system for the failed container, run the **podman mount** command. For example, to view the file system for a failed **keystone** container, run the following command:

   ```
   $ sudo podman mount keystone
   ```

   This provides a mounted location to view the filesystem contents:

   ```
   /var/lib/containers/storage/overlay/78946a109085aeb8b3a350fc20bd8049a08918d74f573396d
   7358270e711c610/merged
   ```

   This is useful for viewing the Puppet reports within the container. You can find these reports in the **var/lib/puppet/** directory within the container mount.

Exporting a container

When a container fails, you might need to investigate the full contents of the file. In this case, you can export the full file system of a container as a **tar** archive. For example, to export the **keystone** container file system, run the following command:

```
$ sudo podman export keystone -o keystone.tar
```

This command creates the **keystone.tar** archive, which you can extract and explore.

28.8. TROUBLESHOOTING COMPUTE NODE FAILURES

Compute nodes use the Compute service to perform hypervisor-based operations. This means the main diagnosis for Compute nodes revolves around this service.

Procedure

1. Source the **stackrc** file:

   ```
   $ source ~/stackrc
   ```

2. Get the IP address of the Compute node that contains the failure:

   ```
   (undercloud) $ openstack server list
   ```

3. Log in to the node:

   ```
   (undercloud) $ ssh tripleo-admin@192.168.24.60
   ```

4. Change to the root user:

   ```
   $ sudo -i
   ```

5. View the status of the container:

   ```
   $ sudo podman ps -f name=nova_compute
   ```

6. The primary log file for Compute nodes is **/var/log/containers/nova/nova-compute.log**. If issues occur with Compute node communication, use this file to begin the diagnosis.

7. If you perform maintenance on the Compute node, migrate the existing instances from the host to an operational Compute node, then disable the node.

28.9. CREATING AN SOSREPORT

If you need to contact Red Hat for support with Red Hat OpenStack Platform, you might need to generate an **sosreport**. For more information about creating an **sosreport**, see:

- "How to collect all required logs for Red Hat Support to investigate an OpenStack issue"

28.10. 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	**~/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. CHANGING THE SSL/TLS CIPHER RULES FOR HAPROXY

If you enabled SSL/TLS in the undercloud (see Section 7.2, “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.

IMPORTANT

The iRMC user must have the **ADMINISTRATOR** privilege.

pm_addr

The IP address of the iRMC interface.

pm_port (Optional)

The port to use for iRMC operations. The default is 443.

pm_auth_method (Optional)

The authentication method for iRMC operations. Use either **basic** or **digest**. The default is **basic**.

pm_client_timeout (Optional)

Timeout, in seconds, for iRMC operations. The default is 60 seconds.

pm_sensor_method (Optional)

Sensor data retrieval method. Use either **ipmitool** or **scci**. The default is **ipmitool**.

- To enable this driver, add **irmc** to the **enabled_hardware_types** option in your **undercloud.conf** and rerun the **openstack undercloud install** command.

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.