Red Hat OpenStack Platform 10

Director Installation and Usage

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

Last Updated: 2020-10-29

Red Hat OpenStack Platform 10 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

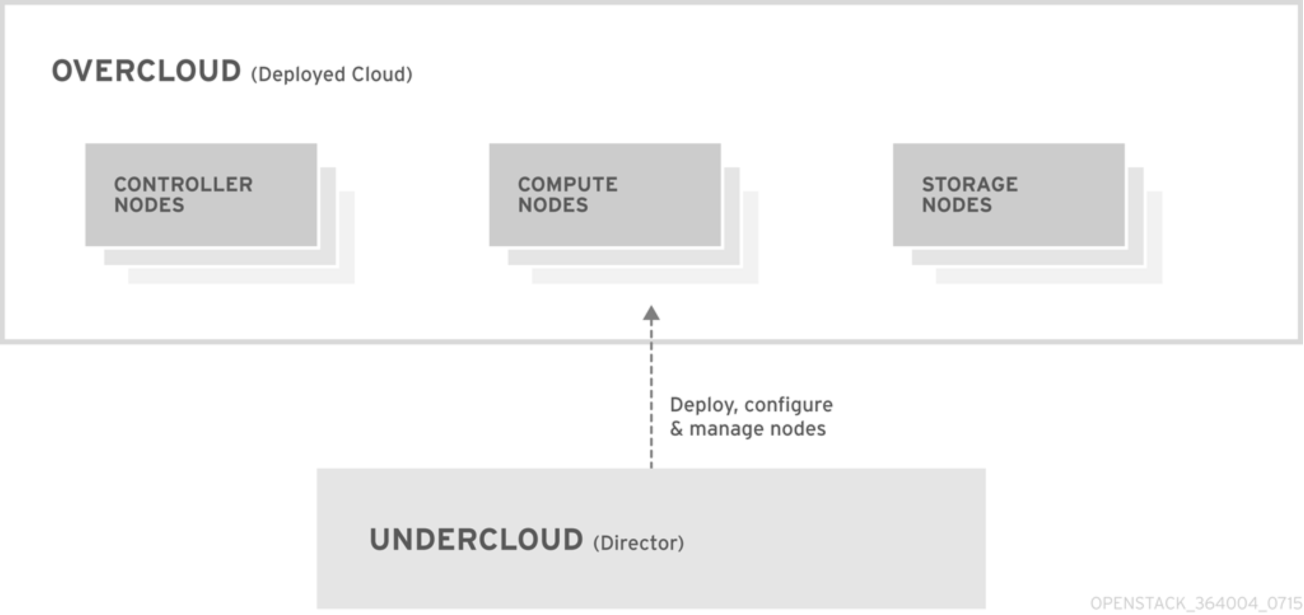
This guide explains how to install Red Hat OpenStack Platform 10 in an enterprise environment using the Red Hat OpenStack Platform director. This includes installing the director, planning your environment, and creating an OpenStack environment with the director.

Table of Contents

CHAPTER 1. INTRODUCTION

The Red Hat OpenStack Platform director is a toolset for installing and managing a complete OpenStack environment. It is based primarily on the OpenStack project TripleO, which is an abbreviation for "OpenStack-On-OpenStack". This project takes advantage of OpenStack components to install a fully operational OpenStack environment. This includes new OpenStack components that provision and control bare metal systems to use as OpenStack nodes. This provides a simple method for installing a complete Red Hat OpenStack Platform environment that is both lean and robust.

The Red Hat OpenStack Platform director uses two main concepts: an undercloud and an overcloud. The undercloud installs and configures the overcloud. The next few sections outline the concept of each.



1.1. UNDERCLOUD

The undercloud is the main director node. 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 provide the multiple functions:

Environment Planning

The undercloud provides planning functions for users to create and assign certain node roles. The undercloud includes a default set of nodes such as Compute, Controller, and various storage roles, but also provides the ability to use custom roles. In addition, you can select which 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 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 to each node. This provides a method to provision bare metal systems as OpenStack nodes. See Appendix B, *Power Management Drivers* for a full list of power management drivers.

Orchestration

The undercloud provides a set of YAML templates that acts as 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 allow you to incorporate your own customizations as certain points in the environment creation process.

Command Line Tools and a Web UI

The Red Hat OpenStack Platform director performs these undercloud functions through a terminal-based command line interface or a web-based user interface.

Undercloud Components

The undercloud uses OpenStack components as its base tool set. This includes the following components:

- OpenStack Identity (keystone) - Provides authentication and authorization for the director's components.
- OpenStack Bare Metal (ironic) and OpenStack Compute (nova) - Manages bare metal nodes.
- OpenStack Networking (neutron) and Open vSwitch - Controls networking for bare metal nodes.
- OpenStack Image Service (glance) - Stores images that are written to bare metal machines.
- OpenStack Orchestration (heat) and Puppet - Provides orchestration of nodes and configuration of nodes after the director writes the overcloud image to disk.
- OpenStack Telemetry (ceilometer) - Performs monitoring and data collection. This also includes:
    - OpenStack Telemetry Metrics (gnocchi) - Provides a time series database for metrics.
    - OpenStack Telemetry Alarming (aodh) - Provides a an alarming component for monitoring.
- OpenStack Workflow Service (mistral) - Provides a set of workflows for certain director-specific actions, such as importing and deploying plans.
- OpenStack Messaging Service (zaqar) - Provides a messaging service for the OpenStack Workflow Service.
- OpenStack Object Storage (swift) - Provides object storage for various OpenStack Platform components, including:
    - Image storage for OpenStack Image Service
    - Introspection data for OpenStack Bare Metal
    - Deployment plans for OpenStack Workflow Service

1.2. OVERCLOUD

The overcloud is the resulting Red Hat OpenStack Platform environment created using the undercloud. This includes different nodes roles which you define based on the OpenStack Platform environment you aim to create. The undercloud includes a default set of overcloud node roles, which include:

Controller

Nodes that provide administration, networking, and high availability for the OpenStack environment. An ideal OpenStack environment recommends three of these nodes together in a high availability cluster.

A default Controller node contains the following components:

- 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 Telemetry (ceilometer)
- OpenStack Telemetry Metrics (gnocchi)
- OpenStack Telemetry Alarming (aodh)
- OpenStack Clustering (sahara)
- OpenStack Shared File Systems (manila)
- OpenStack Bare Metal (ironic)
- MariaDB
- Open vSwitch
- Pacemaker and Galera for high availability services.

Compute

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

- OpenStack Compute (nova)
- KVM/QEMU
- OpenStack Telemetry (ceilometer) agent
- Open vSwitch

Storage

Nodes that provide storage for the OpenStack environment. This includes nodes for:

- Ceph Storage nodes - Used to form storage clusters. Each node contains a Ceph Object Storage Daemon (OSD). In addition, the director installs Ceph Monitor onto the Controller nodes in situations where it deploys Ceph Storage nodes.
- Block storage (cinder) - Used as external block storage for HA Controller nodes. This node contains the following components:
    - OpenStack Block Storage (cinder) volume
    - OpenStack Telemetry (ceilometer) agent
    - Open vSwitch.
- Object storage (swift) - These nodes provide a external storage layer for OpenStack Swift. The Controller nodes access these nodes through the Swift proxy. This node contains the following components:
    - OpenStack Object Storage (swift) storage
    - OpenStack Telemetry (ceilometer) agent
    - Open vSwitch.

1.3. HIGH AVAILABILITY

The Red Hat OpenStack Platform director uses a Controller node cluster to provide high availability services to your OpenStack Platform environment. The director installs a duplicate set of components on each Controller node and manages them 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 Red Hat OpenStack Platform database across the cluster.
- Memcached - Provides database caching.

NOTE

Red Hat OpenStack Platform director automatically configures the bulk of high availability on Controller nodes. However, the nodes require some manual configuration to enable fencing and power management controls. This guide includes these instructions.

1.4. CEPH STORAGE

It is common for large organizations using OpenStack 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/or 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 Red Hat OpenStack Platform 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; this means 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.

See Red Hat Ceph Storage for additional information about Red Hat Ceph Storage.

CHAPTER 2. REQUIREMENTS

This chapter outlines the main requirements for setting up an environment to provision Red Hat OpenStack Platform using the director. This includes the requirements for setting up the director, accessing it, and the hardware requirements for hosts that the director provisions for OpenStack services.

NOTE

Prior to deploying Red Hat OpenStack Platform, it is important to consider the characteristics of the available deployment methods. For more information, refer to the Installing and Managing Red Hat OpenStack Platform .

2.1. ENVIRONMENT REQUIREMENTS

Minimum Requirements:

- 1 host machine for the Red Hat OpenStack Platform director
- 1 host machine for a Red Hat OpenStack Platform Compute node
- 1 host machine for a Red Hat OpenStack Platform Controller node

Recommended Requirements:

- 1 host machine for the Red Hat OpenStack Platform director
- 3 host machines for Red Hat OpenStack Platform Compute nodes
- 3 host machines for Red Hat OpenStack Platform Controller nodes in a cluster
- 3 host machines for Red Hat Ceph Storage nodes in a cluster

Note the following:

- It is recommended to use bare metal systems for all nodes. At minimum, the Compute nodes require bare metal systems.
- All overcloud bare metal systems require an Intelligent Platform Management Interface (IPMI). This is because the director controls the power management.
- Set the each node's internal BIOS clock to UTC. This prevents issues with future-dated file timestamps when **hwclock** synchronizes the BIOS clock before applying the timezone offset.

WARNING

Do not upgrade to the Red Hat Enterprise Linux 7.3 kernel without also upgrading from Open vSwitch (OVS) 2.4.0 to OVS 2.5.0. If only the kernel is upgraded, then OVS will stop functioning.

2.2. UNDERCLOUD REQUIREMENTS

The undercloud system hosting the director provides provisioning and management for all nodes in the overcloud.

- An 8-core 64-bit x86 processor with support for the Intel 64 or AMD64 CPU extensions.
- A minimum of 16 GB of RAM.
- A minimum of 40 GB of available disk space on the root disk. Make sure to leave at least 10 GB free space before attempting an overcloud deployment or update. This free space accommodates image conversion and caching during the node provisioning process.
- A minimum of 2 x 1 Gbps Network Interface Cards. However, it is recommended to use a 10 Gbps interface for Provisioning network traffic, especially if provisioning a large number of nodes in your overcloud environment.
- Red Hat Enterprise Linux 7.7 is installed as the host operating system.
- SELinux is enabled in **Enforcing** mode on the host.

2.2.1. 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 7, as listed on certified hypervisors.
Red Hat Virtualization	Hosted by Red Hat Virtualization 4.x, as listed on certified hypervisors.
Microsoft Hyper-V	Hosted by versions of Hyper-V as listed on the Red Hat Customer Portal Certification Catalogue.
VMware ESX and ESXi	Hosted by versions of ESX and ESXi as listed on the Red Hat Customer Portal Certification Catalogue.

IMPORTANT

Red Hat OpenStack Platform director 10 requires the latest version of Red Hat Enterprise Linux as the host operating system. This means your virtualization platform must also support the underlying Red Hat Enterprise Linux version.

Virtual Machine Requirements

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

Network Considerations

Note the following network considerations for your virtualized undercloud:

Power Management

The undercloud 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 (**ctlplane**) network requires the ability to broadcast and serve DHCP requests to the NICs of the overcloud's bare metal nodes. As a recommendation, create a bridge that connects the VM's NIC to the same network as the bare metal NICs.

> **NOTE**
>
> A common problem occurs when the hypervisor technology blocks the undercloud from transmitting traffic from an unknown address. - If using Red Hat Enterprise Virtualization, disable **anti-mac-spoofing** to prevent this. - If using VMware ESX or ESXi, allow forged transmits to prevent this.

Example Architecture

This is just an example of a basic undercloud virtualization architecture using a KVM server. It is intended as a foundation you can build on depending on your network and resource requirements.

The KVM host uses two Linux bridges:

br-ex (eth0)

- Provides outside access to the undercloud
- DHCP server on outside network assigns network configuration to undercloud using the virtual NIC (eth0)
- Provides access for the undercloud to access the power management interfaces for the bare metal servers

br-ctlplane (eth1)

- Connects to the same network as the bare metal overcloud nodes
- Undercloud fulfills DHCP and PXE boot requests through virtual NIC (eth1)
- Bare metal servers for the overcloud boot through PXE over this network

The KVM host requires the following packages:

```
$ yum install libvirt-client libvirt-daemon qemu-kvm libvirt-daemon-driver-qemu libvirt-daemon-kvm
virt-install bridge-utils rsync
```

The following command creates the undercloud virtual machine on the KVM host and create two virtual NICs that connect to the respective bridges:

```
$ virt-install --name undercloud --memory=16384 --vcpus=4 --location /var/lib/libvirt/images/rhel-
server-7.3-x86_64-dvd.iso --disk size=100 --network bridge=br-ex --network bridge=br-ctlplane --
graphics=vnc --hvm --os-variant=rhel7
```

This starts a **libvirt** domain. Connect to it with **virt-manager** and walk through the install process. Alternatively, you can perform an unattended installation using the following options to include a kickstart file:

```
--initrd-inject=/root/ks.cfg --extra-args "ks=file:/ks.cfg"
```

Once installation completes, SSH into the instance as the **root** user and follow the instructions in Chapter 4, *Installing the Undercloud*

Backups

To back up a virtualized undercloud, there are multiple solutions:

- **Option 1:** Follow the instructions in the Back Up and Restore the Director Undercloud Guide.
- **Option 2:** Shut down the undercloud and take a copy of the undercloud virtual machine storage backing.
- **Option 3:** Take a snapshot of the undercloud VM if your hypervisor supports live or atomic snapshots.

If using a KVM server, use the following procedure to take a snapshot:

1. Make sure **qemu-guest-agent** is running on the undercloud guest VM.
2. Create a live snapshot of the running VM:

```
$ virsh snapshot-create-as --domain undercloud --disk-only --atomic --quiesce
```

1. Take a copy of the (now read-only) QCOW backing file

```
$ rsync --sparse -avh --progress /var/lib/libvirt/images/undercloud.qcow2 1.qcow2
```

1. Merge the QCOW overlay file into the backing file and switch the undercloud VM back to using the original file:

```
$ virsh blockcommit undercloud vda --active --verbose --pivot
```

2.3. NETWORKING REQUIREMENTS

The undercloud host requires at least two networks:

- Provisioning network - Provides DHCP and PXE boot functions to help discover bare metal systems for use in the overcloud. Typically, this network must use a native VLAN on a trunked interface so that the director serves PXE boot and DHCP requests. Some server hardware BIOSes support PXE boot from a VLAN, but the BIOS must also support translating that VLAN into a native VLAN after booting, otherwise the undercloud will not be reachable. Currently, only a small subset of server hardware fully supports this feature. This is also the network you use to control power management through Intelligent Platform Management Interface (IPMI) on all overcloud nodes.
- External Network - A separate network for external access to the overcloud and undercloud. The interface connecting to this network requires a routable IP address, either defined statically, or dynamically through an external DHCP service.

This represents the minimum number of networks required. However, the director can isolate other Red Hat OpenStack Platform network traffic into other networks. Red Hat OpenStack Platform supports both physical interfaces and tagged VLANs for network isolation.

Note the following:

- Typical minimal overcloud network configuration can include:
  - 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.
  - Dual NIC configuration - One NIC for the Provisioning network and the other NIC for the External network.
  - Dual NIC configuration - One NIC for the Provisioning network on the native VLAN and the other NIC for tagged VLANs that use subnets for the different overcloud network types.
  - Multiple NIC configuration - Each NIC uses a subnet for a different overcloud network type.
- Additional physical NICs can be used for isolating individual networks, creating bonded interfaces, or for delegating tagged VLAN traffic.
- If using VLANs to isolate your network traffic types, use a switch that supports 802.1Q standards to provide tagged VLANs.
- During the overcloud creation, you will 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.
- Make sure the Provisioning network NIC is not the same NIC used for remote connectivity on the director machine. The director installation creates a bridge 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 IP address per node connected to the Provisioning network.
  - If planning a high availability configuration, include an extra IP address for the virtual IP of the cluster.
  - Include additional IP addresses within the range for scaling the environment.

NOTE

Duplicate IP addresses should be avoided on the Provisioning network. For more information, see Section 3.2, “Planning Networks”.

NOTE

For more information on planning your IP address usage, for example, for storage, provider, and tenant networks, see the Networking Guide.

- 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). This allows the director to 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 will be useful later when setting up the overcloud nodes.

- Fencing is mandatory for Red Hat support. You must add a network flow from controller to OOB IPMI.

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

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

- It is recommended to use 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.

IMPORTANT

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

- 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.

- Ensure proper firewall rules and password usage.

- Ensure that SELinux is enabled.

For details on securing your system, see:

- Red Hat Enterprise Linux 7 Security Guide

- Red Hat Enterprise Linux 7 SELinux User's and Administrator's Guide

2.4. OVERCLOUD REQUIREMENTS

The following sections detail the requirements for individual systems and nodes in the overcloud installation.

NOTE

Booting an overcloud node from the SAN (FC-AL, FCoE, iSCSI) is not yet supported.

2.4.1. Compute Node Requirements

Compute nodes are responsible for running virtual machine instances after they are launched. Compute nodes must support hardware virtualization. Compute nodes must also have enough memory and disk space to support the requirements of the virtual machine instances 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 this processor has a minimum of 4 cores.

Memory

A minimum of 6 GB of RAM. + Add additional RAM to this requirement based on the amount of memory that you intend to make available to virtual machine instances.

Disk Space

A minimum of 40 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 Controller node requires a supported power management interface, such as an Intelligent Platform Management Interface (IPMI) functionality, on the server's motherboard.

2.4.2. Controller Node Requirements

Controller nodes are responsible for hosting the core services in a RHEL OpenStack Platform environment, such as the Horizon dashboard, the back-end database server, Keystone authentication, and High Availability services.

Processor

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

Memory

Minimum amount of memory is 32 GB. However, the amount of recommended memory depends on the number of vCPUs (which is based on CPU cores multiplied by hyper-threading value). Use the following calculations as guidance:

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

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

Disk Storage and Layout

By default, the Telemetry (**gnocchi**) and Object Storage (**swift**) services are both installed on the Controller, with both configured to use the root disk. These defaults are suitable for deploying small overclouds built on commodity hardware; such environments are typical of proof-of-concept and test environments. These defaults also allow the deployment of overclouds with minimal planning but offer little in terms of workload capacity and performance.
In an enterprise environment, however, this could cause a significant bottleneck, as 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 need to plan your overcloud and configure it accordingly.

Red Hat provides several configuration recommendations for both Telemetry and Object Storage. See Deployment Recommendations for Specific Red Hat OpenStack Platform Services for details.

Disk Space

A minimum of 40 GB of available disk space, if the Object Storage service (swift) is not running on the controller nodes.

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's motherboard.

2.4.3. Ceph Storage Node Requirements

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

Placement Groups

Ceph uses Placement Groups to facilitate dynamic and efficient object tracking at scale. In the case of OSD failure or cluster re-balancing, Ceph can move or replicate a placement group and its contents, which means a Ceph cluster can re-balance and recover efficiently. The default Placement Group count that Director creates is not always optimal so it is important to calculate the correct Placement Group count according to your requirements. You can use the Placement Group calculator to calculate the correct count: Ceph Placement Groups (PGs) per Pool Calculator

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. Ideally, use at minimum 1 GB of memory per 1 TB of hard disk space.

Disk Space

Storage requirements depends on the amount of memory. Ideally, use at minimum 1 GB of memory per 1 TB of hard disk space.

Disk Layout

The recommended Red Hat Ceph Storage node configuration requires at least three or more disks in a layout similar to the following:

- **/dev/sda** - The root disk. The director copies the main Overcloud image to the disk.

- **/dev/sdb** - The journal disk. This disk divides into partitions for Ceph OSD journals. For example, **/dev/sdb1**, **/dev/sdb2**, **/dev/sdb3**, and onward. The journal disk is usually a solid state drive (SSD) to aid with system performance.
- **/dev/sdc** and onward - The OSD disks. Use as many disks as necessary for your storage requirements.

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. It is recommended to use a 10 Gbps interface for storage node, especially if creating an OpenStack Platform environment that serves a high volume of traffic.

Power Management

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

See the Red Hat Ceph Storage for the Overcloud guide for more information about installing an overcloud with a Ceph Storage cluster.

2.4.4. Object Storage Node Requirements

Object Storage nodes provides an object storage layer for the overcloud. The Object Storage proxy is installed on Controller nodes. The storage layer will require bare metal nodes with multiple number of disks per 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. Ideally, use at minimum 1 GB of memory per 1 TB of hard disk space. For optimal performance, it is recommended to use 2 GB per 1 TB of hard disk space, especially for small file (less 100GB) workloads.

Disk Space

Storage requirements depends 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 of about 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 STORING 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:

- **/dev/sda** - The root disk. The 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's motherboard.

2.5. REPOSITORY REQUIREMENTS

Both the undercloud and overcloud require access to Red Hat repositories either through the Red Hat Content Delivery Network, or through Red Hat Satellite 5 or 6. If using a Red Hat Satellite Server, synchronize the required repositories to your OpenStack Platform environment. Use the following list of CDN channel names as a guide:

WARNING

Do not upgrade to the Red Hat Enterprise Linux 7.3 kernel without also upgrading from Open vSwitch (OVS) 2.4.0 to OVS 2.5.0. If only the kernel is upgraded, then OVS will stop functioning.

Table 2.1. OpenStack Platform Repositories

Name	Repository	Description of Requirement
Red Hat Enterprise Linux 7 Server (RPMs)	**rhel-7-server-rpms**	Base operating system repository.
Red Hat Enterprise Linux 7 Server - Extras (RPMs)	**rhel-7-server-extras-rpms**	Contains Red Hat OpenStack Platform dependencies.
Red Hat Enterprise Linux 7 Server - RH Common (RPMs)	**rhel-7-server-rh-common-rpms**	Contains tools for deploying and configuring Red Hat OpenStack Platform.
Red Hat Satellite Tools for RHEL 7 Server RPMs x86_64	**rhel-7-server-satellite-tools-6.2-rpms**	Tools for managing hosts with Red Hat Satellite 6.
Red Hat Enterprise Linux High Availability (for RHEL 7 Server) (RPMs)	**rhel-ha-for-rhel-7-server-rpms**	High availability tools for Red Hat Enterprise Linux. Used for Controller node high availability.
Red Hat Enterprise Linux OpenStack Platform 10 for RHEL 7 (RPMs)	**rhel-7-server-openstack-10-rpms**	Core Red Hat OpenStack Platform repository. Also contains packages for Red Hat OpenStack Platform director.

Red Hat Ceph Storage OSD 2 for Red Hat Enterprise Linux 7 Server (RPMs)	**rhel-7-server-rhceph-2-osd-rpms**	(For Ceph Storage Nodes) Repository for Ceph Storage Object Storage daemon. Installed on Ceph Storage nodes.
Red Hat Ceph Storage MON 2 for Red Hat Enterprise Linux 7 Server (RPMs)	**rhel-7-server-rhceph-2-mon-rpms**	(For Ceph Storage Nodes) Repository for Ceph Storage Monitor daemon. Installed on Controller nodes in OpenStack environments using Ceph Storage nodes.
Red Hat Ceph Storage Tools 2 for Red Hat Enterprise Linux 7 Server (RPMs)	**rhel-7-server-rhceph-2-tools-rpms**	Provides tools for nodes to communicate with the Ceph Storage cluster. This repository should be enabled for all overcloud nodes when deploying an overcloud with a Ceph Storage cluster.

NOTE

To configure repositories for your Red Hat OpenStack Platform environment in an offline network, see "Configuring Red Hat OpenStack Platform Director in an Offline Environment" on the Red Hat Customer Portal.

CHAPTER 3. PLANNING YOUR OVERCLOUD

The following section provides some guidelines on planning various aspects of your Red Hat OpenStack Platform environment. This includes defining node roles, planning your network topology, and storage.

3.1. PLANNING NODE DEPLOYMENT ROLES

The director provides multiple default node types for building your overcloud. These node types are:

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 environment requires three Controller nodes for a highly available environment.

NOTE

Environments with one node can be used for testing purposes. Environments with two nodes or more than three nodes are not supported.

Compute

A physical server that acts as a hypervisor, and provides the processing capabilities required for running virtual machines in the environment. A basic Red Hat OpenStack Platform 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 for OpenStack's swift service. This deployment role is optional.

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

Table 3.1. 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 on the composable roles architecture, see Chapter 8. Composable Roles and Services in the *Advanced Overcloud Customization* guide.

3.2. PLANNING NETWORKS

It is important to plan your environment's networking topology and subnets so that you can properly map roles and services to correctly communicate with each other. Red Hat OpenStack Platform uses the neutron networking service, which operates autonomously and manages software-based networks, static and floating IP addresses, and DHCP. The director deploys this service on each Controller node in an overcloud environment.

Red Hat OpenStack Platform maps the different services onto separate network traffic types, which are assigned to the various subnets in your environments. These network traffic types include:

Table 3.2. Network Type Assignments

Network Type	Description	Used By
IPMI	Network used for power management of nodes. This network is predefined before the installation of the undercloud.	All nodes
Provisioning	The director uses this network traffic type to deploy new nodes over PXE boot and orchestrate the installation of OpenStack Platform on the overcloud bare metal servers. This network is predefined before the installation of the undercloud.	All nodes
Internal API	The Internal API network is used for communication between the OpenStack services using API communication, RPC messages, and database communication.	Controller, Compute, Cinder Storage, Swift Storage

Tenant	Neutron provides each tenant with their own networks using either VLAN segregation (where each tenant network is a network VLAN), or 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.	Controller, Compute
Storage	Block Storage, NFS, iSCSI, and others. Ideally, this would be isolated to an entirely separate switch fabric for performance reasons.	All nodes
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 back end 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.	Controller, Ceph Storage, Cinder Storage, Swift Storage
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. If the external network uses private IP addresses (as per RFC-1918), then further NAT must be performed for traffic originating from the internet.	Controller and undercloud

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 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, neutron creates an OVS port with the VLAN segmentation ID on the chosen bridge for each Floating IP network.	Controller
Management	Provides access for system administration functions such as SSH access, DNS traffic, and NTP traffic. This network also acts as a gateway for non-Controller nodes	All nodes

In a typical Red Hat OpenStack Platform installation, the number of network types often exceeds the number of physical network links. In order to connect all the networks to the proper hosts, the overcloud uses VLAN tagging to deliver more than one network per interface. Most of the networks are isolated subnets but some require a Layer 3 gateway to provide routing for Internet access or infrastructure network connectivity.

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.

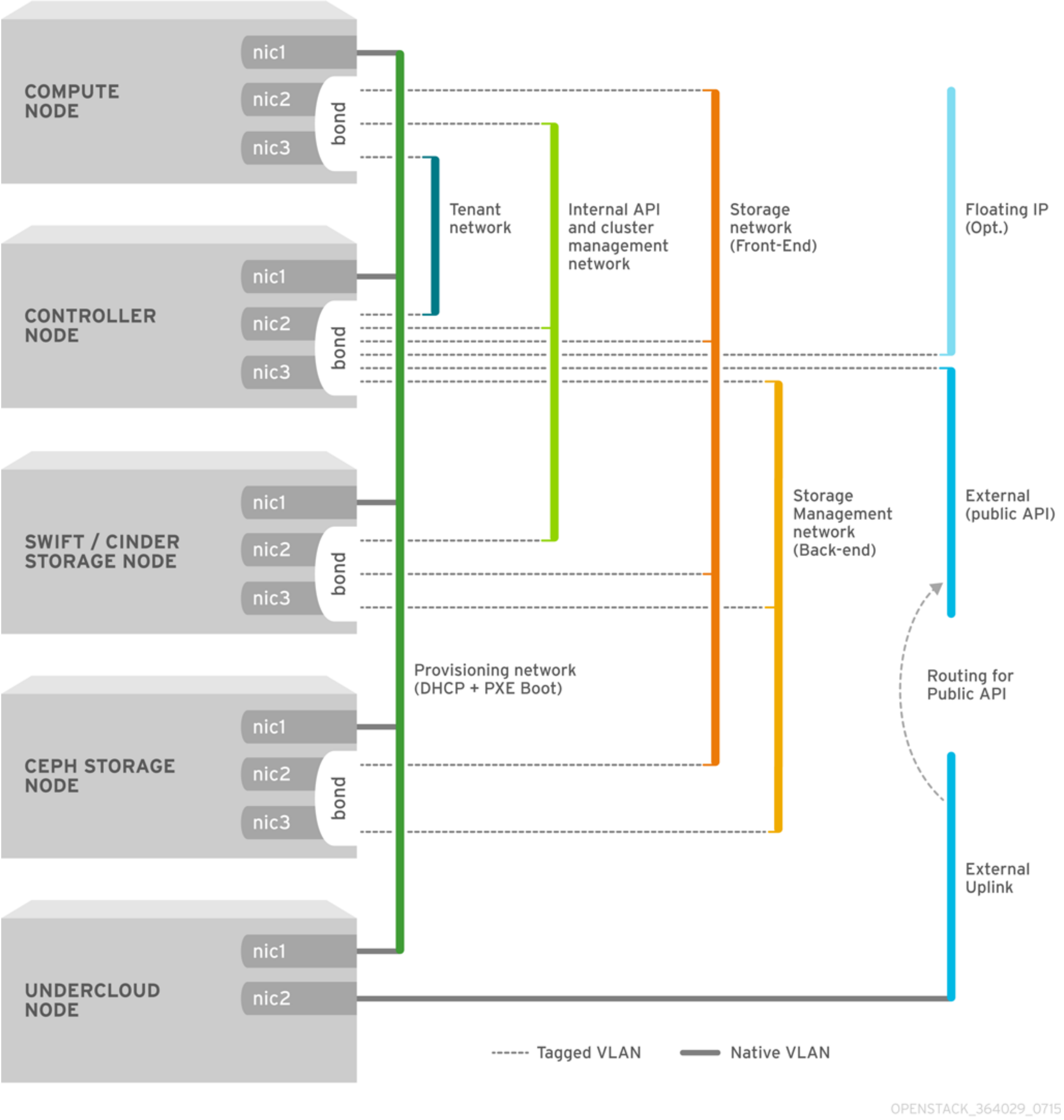
The director provides a method for mapping six of these traffic types to certain subnets or VLANs. These traffic types include:

- Internal API
- Storage
- Storage Management

- Tenant Networks
- External
- Management (optional)

Any unassigned networks are automatically assigned to the same subnet as the Provisioning network.

The diagram below provides an example of a network topology where the networks are isolated on separate VLANs. Each overcloud node uses two interfaces (**nic2** and **nic3**) in a bond to deliver these networks over their respective VLANs. Meanwhile, each overcloud node communicates with the undercloud over the Provisioning network through a native VLAN using **nic1**.



The following table provides examples of network traffic mappings different network layouts:

Table 3.3. Network Mappings

	Mappings	Total Interfaces	Total VLANs
Flat Network with External Access	Network 1 - Provisioning, Internal API, Storage, Storage Management, Tenant Networks Network 2 - External, Floating IP (mapped after overcloud creation)	2	2
Isolated Networks	Network 1 - Provisioning Network 2 - Internal API Network 3 - Tenant Networks Network 4 - Storage Network 5 - Storage Management Network 6 - Management (optional) Network 7 - External, Floating IP (mapped after overcloud creation)	3 (includes 2 bonded interfaces)	7

3.3. PLANNING STORAGE

NOTE

Using LVM on a guest instance that uses a back end cinder-volume of any driver or back-end type results in issues with performance, volume visibility and availability, and data corruption. These issues can be mitigated using a LVM filter. For more information, refer to section 2.1 Back Ends in the *Storage Guide* and KCS article 3213311, "Using LVM on a cinder volume exposes the data to the compute host."

The director provides different storage options for the overcloud environment. This includes:

Ceph Storage Nodes

The director creates a set of scalable storage nodes using Red Hat Ceph Storage. The overcloud uses these nodes for:

- **Images** - Glance manages images for VMs. Images are immutable. OpenStack treats images as binary blobs and downloads them accordingly. You can use glance to store images in a Ceph Block Device.

- **Volumes** - Cinder volumes are block devices. OpenStack uses volumes to boot VMs, or to attach volumes to running VMs. OpenStack manages volumes using Cinder services. You can use Cinder to boot a VM using a copy-on-write clone of an image.

- **Guest Disks** - Guest disks are guest operating system disks. By default, when you boot a virtual machine with nova, its disk appears as a file on the filesystem of the hypervisor (usually under **/var/lib/nova/instances/<uuid>/**). It is possible to boot every virtual machine inside Ceph directly without using cinder, which is advantageous because it allows you to perform maintenance operations easily with the live-migration process. Additionally, if your hypervisor dies it is also convenient to trigger **nova evacuate** and run the virtual machine elsewhere almost seamlessly.

IMPORTANT

If you want to boot virtual machines in Ceph (ephemeral back end or boot from volume), the glance image format must be **RAW** format. Ceph does not support other image formats such as QCOW2 or VMDK for hosting a virtual machine disk.

See Red Hat Ceph Storage Architecture Guide for additional information.

Swift Storage Nodes

The director creates an external object storage node. This is useful in situations where you need to scale or replace controller nodes in your overcloud environment but need to retain object storage outside of a high availability cluster.

CHAPTER 4. INSTALLING THE UNDERCLOUD

The first step to creating your Red Hat OpenStack Platform environment is to install the director on the undercloud system. This involves a few prerequisite steps to enable the necessary subscriptions and repositories.

4.1. CREATING A DIRECTOR INSTALLATION USER

The director installation process requires a non-root user to execute commands. Use the following commands to create the user named **stack** and set a password:

```
[root@director ~]# useradd stack
[root@director ~]# passwd stack  # specify a password
```

Disable password requirements for this user 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
```

Switch to the new **stack** user:

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

Continue the director installation as the **stack** user.

4.2. CREATING DIRECTORIES FOR TEMPLATES AND IMAGES

The director uses system images and Heat templates to create the overcloud environment. To keep these files organized, we recommend creating directories for images and templates:

```
$ mkdir ~/images
$ mkdir ~/templates
```

Other sections in this guide use these two directories to store certain files.

4.3. SETTING THE HOSTNAME FOR THE SYSTEM

The director requires a fully qualified domain name for its installation and configuration process. This means you may need to set the hostname of your director’s host. Check the hostname of your host:

```
$ hostname    # Checks the base hostname
$ hostname -f # Checks the long hostname (FQDN)
```

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

```
$ sudo hostnamectl set-hostname manager.example.com
$ sudo hostnamectl set-hostname --transient manager.example.com
```

The director also requires an entry for the system's hostname and base name in **/etc/hosts**. For example, if the system is named **manager.example.com**, then **/etc/hosts** requires an entry like:

```
127.0.0.1   manager.example.com manager localhost localhost.localdomain localhost4
localhost4.localdomain4
```

4.4. REGISTERING YOUR SYSTEM

To install the Red Hat OpenStack Platform director, first register the host system using Red Hat Subscription Manager, and subscribe to the required channels.

Register your system with the Content Delivery Network, entering your Customer Portal user name and password when prompted:

```
$ sudo subscription-manager register
```

Find the entitlement pool ID for Red Hat OpenStack Platform director. For example:

```
$ sudo subscription-manager list --available --all --matches="*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
```

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

```
$ sudo subscription-manager attach --pool=Valid-Pool-Number-123456
```

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

```
$ sudo subscription-manager repos --disable=*
$ sudo subscription-manager repos --enable=rhel-7-server-rpms --enable=rhel-7-server-extras-rpms -
-enable=rhel-7-server-rh-common-rpms --enable=rhel-ha-for-rhel-7-server-rpms --enable=rhel-7-
server-openstack-10-rpms
```

These repositories contain packages the director installation requires.

IMPORTANT

Only enable the repositories listed in Section 2.5, "Repository Requirements". Additional repositories can cause package and software conflicts. Do not enable any additional repositories.

Set the RHEL version to RHEL 7.7:

```
$ sudo subscription-manager release --set=7.7
```

Perform an update on your system to make sure you have the latest base system packages:

```
$ sudo yum update -y
$ sudo reboot
```

The system is now ready for the director installation.

4.5. INSTALLING THE DIRECTOR PACKAGES

Use the following command to install the required command line tools for director installation and configuration:

```
$ sudo yum install -y python-tripleoclient
```

This installs all packages required for the director installation.

4.6. CONFIGURING THE DIRECTOR

The director installation process requires certain settings to determine your network configurations. The settings are stored in a template located in the **stack** user's home directory as **undercloud.conf**.

Red Hat provides a basic template to help determine the required settings for your installation. Copy this template to the **stack** user's home directory:

```
$ cp /usr/share/instack-undercloud/undercloud.conf.sample ~/undercloud.conf
```

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

The template contains two sections: **[DEFAULT]** and **[auth]**. The **[DEFAULT]** section contains the following parameters:

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 the user must configure all system host name settings appropriately.

local_ip

The IP address defined for the director's Provisioning NIC. This is also the IP address the director uses for its DHCP and PXE boot services. Leave this value as the default **192.0.2.1/24** unless you are using a different subnet for the Provisioning network, for example, if it conflicts with an existing IP address or subnet in your environment.

network_gateway

The gateway for the overcloud instances. This is the undercloud host, which forwards traffic to the External network. Leave this as the default **192.0.2.1** unless you are either using a different IP address for the director or want to directly use an external gateway.

NOTE

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

undercloud_public_vip

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.

undercloud_admin_vip

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.

undercloud_service_certificate

The location and filename of the certificate for OpenStack SSL communication. Ideally, you obtain this certificate from a trusted certificate authority. Otherwise generate your own self-signed certificate using the guidelines in Appendix A, *SSL/TLS Certificate Configuration*. These guidelines also contain instructions on setting the SELinux context for your certificate, whether self-signed or from an authority.

generate_service_certificate

Defines whether to generate an SSL 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.

certificate_generation_ca

The **certmonger** nickname of the CA that signs the requested certificate. Only use this option 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 it to the trust chain.

service_principal

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

local_interface

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

```
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen
1000
   link/ether 52:54:00:75:24:09 brd ff:ff:ff:ff:ff:ff
   inet 192.168.122.178/24 brd 192.168.122.255 scope global dynamic eth0
     valid_lft 3462sec preferred_lft 3462sec
   inet6 fe80::5054:ff:fe75:2409/64 scope link
     valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noop state DOWN
   link/ether 42:0b:c2:a5:c1:26 brd ff:ff:ff:ff:ff:ff
```

In this example, the External NIC uses **eth0** and the Provisioning NIC uses **eth1**, which is currently not configured. In this case, set the **local_interface** to **eth1**. The configuration script attaches this interface to a custom bridge defined with the **inspection_interface** parameter.

local_mtu

MTU to use for the **local_interface**.

network_cidr

The network that the director uses to manage overcloud instances. This is the Provisioning network, which the undercloud's **neutron** service manages. Leave this as the default **192.0.2.0/24** unless you are using a different subnet for the Provisioning network.

masquerade_network

Defines the network that will masquerade for external access. This provides the Provisioning network with a degree of network address translation (NAT) so that it has external access through the director. Leave this as the default (**192.0.2.0/24**) unless you are using a different subnet for the Provisioning network.

dhcp_start; dhcp_end

The start and end of the DHCP allocation range for overcloud nodes. Ensure this range contains enough IP addresses to allocate your nodes.

hieradata_override

Path to **hieradata** override file. If set, the undercloud installation copies this file under **/etc/puppet/hieradata** and sets it as the first file in the hierarchy. Use this to provide custom configuration to services beyond the **undercloud.conf** parameters.

net_config_override

Path to network configuration override template. If set, the undercloud uses a JSON format template to configure the networking with **os-net-config**. This ignores the network parameters set in **undercloud.conf**. See **/usr/share/instack-undercloud/templates/net-config.json.template** for an example.

inspection_interface

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

inspection_iprange

A range of IP address that the director's introspection service uses during the PXE boot and provisioning process. Use comma-separated values to define the start and end of this range. For example, **192.0.2.100,192.0.2.120**. Make sure this range contains enough IP addresses for your nodes and does not conflict with the range for **dhcp_start** and **dhcp_end**.

inspection_extras

Defines whether to enable extra hardware collection during the inspection process. Requires **python-hardware** or **python-hardware-detect** package on the introspection image.

inspection_runbench

Runs a set of benchmarks during node introspection. Set to **true** to enable. This option is necessary if you intend to perform benchmark analysis when inspecting the hardware of registered nodes. See Section 5.2, "Inspecting the Hardware of Nodes" for more details.

inspection_enable_uefi

Defines whether to support introspection of nodes with UEFI-only firmware. For more information, see Appendix D, *Alternative Boot Modes* .

undercloud_debug

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

enable_tempest

Defines whether to install the validation tools. The default is set to **false**, but you can can enable using **true**.

enable_mistral

Defines whether to install the OpenStack Workflow Service (mistral) in the undercloud.

enable_zaqar

Defines whether to install the OpenStack Messaging Service (zaqar) in the undercloud.

ipxe_enabled

Defines whether to use iPXE or standard PXE. The default is **true**, which enables iPXE. Set to **false** to set to standard PXE. For more information, see Appendix D, *Alternative Boot Modes* .

enable_telemetry

Defines whether to install OpenStack Telemetry (ceilometer, aodh) services in the undercloud. The default value is **false**, which disables telemetry on the undercloud. This parameter is required if using other products that consume metrics data, such as Red Hat CloudForms.

enable_ui

Defines Whether to install the director's web UI. This allows you to perform overcloud planning and deployments through a graphical web interface. For more information, see Chapter 6, *Configuring Basic Overcloud Requirements with the Web UI*. Note that the UI is only available with SSL/TLS enabled using either the **undercloud_service_certificate** or **generate_service_certificate**.

enable_validations

Defines whether to install the requirements to run validations.

store_events

Defines whether to store events in OpenStack Telemetry (ceilometer) on the undercloud.

scheduler_max_attempts

Maximum number of times the scheduler attempts to deploy an instance. Keep this greater or equal to the number of bare metal nodes you expect to deploy at once to work around potential race condition when scheduling.

The **[auth]** section contains the following parameters:

undercloud_db_password; undercloud_admin_token; undercloud_admin_password; undercloud_glance_password; etc

The remaining parameters are the access details for all of the director's services. No change is required for the values. The director's configuration script automatically generates these values if blank in **undercloud.conf**. You can retrieve all values after the configuration script completes.

IMPORTANT

The configuration file examples for these parameters use **<None>** as a placeholder value. Setting these values to **<None>** leads to a deployment error.

Modify the values for these parameters to suit your network. When complete, save the file and run the following command:

```
$ openstack undercloud install
```

This launches the director's configuration script. The director installs additional packages and configures its services to suit the settings in the **undercloud.conf**. This script takes several minutes to complete.

The configuration script generates two files when complete:

- **undercloud-passwords.conf** - A list of all passwords for the director's services.
- **stackrc** - A set of initialization variables to help you access the director's command line tools.

The configuration also starts all OpenStack Platform services automatically. Check the enabled services using the following command:

```
$ sudo systemctl list-units openstack-*
```

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

```
$ source ~/stackrc
```

You can now use the director's command line tools.

4.7. OBTAINING IMAGES FOR OVERCLOUD NODES

The director requires several disk images for provisioning overcloud nodes. This includes:

- An introspection kernel and ramdisk - Used for bare metal system introspection over PXE boot.
- A deployment kernel and ramdisk - Used for system provisioning and deployment.
- An overcloud kernel, ramdisk, and full image - A base overcloud system that is written to the node's hard disk.

Obtain these images from the **rhosp-director-images** and **rhosp-director-images-ipa** packages:

```
$ sudo yum install rhosp-director-images rhosp-director-images-ipa
```

Extract the archives to the **images** directory on the **stack** user's home (**/home/stack/images**):

```
$ cd ~/images
$ for i in /usr/share/rhosp-director-images/overcloud-full-latest-10.0.tar /usr/share/rhosp-director-images/ironic-python-agent-latest-10.0.tar; do tar -xvf $i; done
```

Import these images into the director:

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

This uploads the following images into the director: **bm-deploy-kernel**, **bm-deploy-ramdisk**, **overcloud-full**, **overcloud-full-initrd**, **overcloud-full-vmlinuz**. These are the images for deployment and the overcloud. The script also installs the introspection images on the director's PXE server.

View a list of the images in the CLI:

```
$ openstack image list
+--------------------------------------+------------------------+
| ID                                   | Name                   |
+--------------------------------------+------------------------+
| 765a46af-4417-4592-91e5-a300ead3faf6 | bm-deploy-ramdisk      |
| 09b40e3d-0382-4925-a356-3a4b4f36b514 | bm-deploy-kernel       |
| ef793cd0-e65c-456a-a675-63cd57610bd5 | overcloud-full         |
| 9a51a6cb-4670-40de-b64b-b70f4dd44152 | overcloud-full-initrd  |
| 4f7e33f4-d617-47c1-b36f-cbe90f132e5d | overcloud-full-vmlinuz |
+--------------------------------------+------------------------+
```

This list will not show the introspection PXE images. The director copies these files to **/httpboot**.

```
[stack@host1 ~]$ ls -l /httpboot
total 341460
-rwxr-xr-x. 1 root root   5153184 Mar 31 06:58 agent.kernel
-rw-r--r--. 1 root root 344491465 Mar 31 06:59 agent.ramdisk
-rw-r--r--. 1 root root       337 Mar 31 06:23 inspector.ipxe
```

NOTE

The default **overcloud-full.qcow2** image is a flat partition image. However, you can also import and use whole disk images. See *Appendix C, Whole Disk Images* for more information.

4.8. SETTING A NAMESERVER ON THE UNDERCLOUD'S NEUTRON SUBNET

If you intend for the overcloud to resolve external hostnames, such as **cdn.redhat.com**, it is recommended to set a nameserver on the overcloud nodes. For a standard overcloud without network isolation, the nameserver is defined using the undercloud's **neutron** subnet. Use the following commands to define nameservers for the environment:

```
$ openstack subnet list
$ openstack subnet set --dns-nameserver [nameserver1-ip] --dns-nameserver [nameserver2-ip]
[subnet-uuid]
```

View the subnet to verify the nameserver:

```
$ openstack subnet show [subnet-uuid]
+-------------------+-----------------------------------------------+
| Field             | Value                                         |
+-------------------+-----------------------------------------------+
| ...               |                                               |
| dns_nameservers   | 8.8.8.8                                       |
| ...               |                                               |
+-------------------+-----------------------------------------------+
```

IMPORTANT

If you aim to isolate service traffic onto separate networks, the overcloud nodes use the **DnsServer** parameter in your network environment files.

4.9. BACKING UP THE UNDERCLOUD

Red Hat provides a process to back up important data from the undercloud host and the Red Hat OpenStack Platform director. For more information about undercloud backups, see the "Back Up and Restore the Director Undercloud" guide.

4.10. COMPLETING THE UNDERCLOUD CONFIGURATION

This completes the undercloud configuration. The next chapter explores basic overcloud configuration, including registering nodes, inspecting them, and then tagging them into various node roles.

CHAPTER 5. CONFIGURING BASIC OVERCLOUD REQUIREMENTS WITH THE CLI TOOLS

This chapter provides the basic configuration steps for an OpenStack Platform environment using the CLI tools. An overcloud with a basic configuration contains no custom features. However, you can add advanced configuration options to this basic overcloud and customize it to your specifications using the instructions in the Advanced Overcloud Customization guide.

For the examples in this chapter, all nodes in this chapter are bare metal systems using IPMI for power management. For more supported power management types and their options, see Appendix B, *Power Management Drivers*.

Workflow

1. Create a node definition template and register blank nodes in the director.
2. Inspect hardware of all nodes.
3. Tag nodes into roles.
4. Define additional node properties.

Requirements

- The director node created in Chapter 4, *Installing the Undercloud*
- A set of bare metal machines for your nodes. The number of node required depends on the type of overcloud you intend to create (see Section 3.1, "Planning Node Deployment Roles" for information on overcloud roles). These machines also must comply with the requirements set for each node type. For these requirements, see Section 2.4, "Overcloud Requirements". These nodes do not require an operating system. The director copies a Red Hat Enterprise Linux 7 image to each node.
- One network connection for our Provisioning network, which is configured as a native VLAN. All nodes must connect to this network and comply with the requirements set in Section 2.3, "Networking Requirements". For the examples in this chapter, we use 192.0.2.0/24 as the Provisioning subnet with the following IP address assignments:

  Table 5.1. Provisioning Network IP Assignments

Node Name	IP Address	MAC Address	IPMI IP Address
Director	192.0.2.1	aa:aa:aa:aa:aa:aa	None required
Controller	DHCP defined	bb:bb:bb:bb:bb:bb	192.0.2.205
Compute	DHCP defined	cc:cc:cc:cc:cc:cc	192.0.2.206

- All other network types use the Provisioning network for OpenStack services. However, you can create additional networks for other network traffic types.

5.1. REGISTERING NODES FOR THE OVERCLOUD

The director requires a node definition template, which you create manually. This file (**instackenv.json**) uses the JSON format file, and contains the hardware and power management details for your nodes. For example, a template for registering two nodes might look like this:

```
{
    "nodes":[
        {
            "mac":[
                "bb:bb:bb:bb:bb:bb"
            ],
            "name":"node01",
            "cpu":"4",
            "memory":"6144",
            "disk":"40",
            "arch":"x86_64",
            "pm_type":"pxe_ipmitool",
            "pm_user":"admin",
            "pm_password":"p@55w0rd!",
            "pm_addr":"192.0.2.205"
        },
        {
            "mac":[
                "cc:cc:cc:cc:cc:cc"
            ],
            "name":"node02",
            "cpu":"4",
            "memory":"6144",
            "disk":"40",
            "arch":"x86_64",
            "pm_type":"pxe_ipmitool",
            "pm_user":"admin",
            "pm_password":"p@55w0rd!",
            "pm_addr":"192.0.2.206"
        }
    ]
}
```

This template uses the following attributes:

name

The logical name for the node.

pm_type

The power management driver to use. This example uses the IPMI driver (**pxe_ipmitool**), which is the preferred driver for power management.

pm_user; pm_password

The IPMI username and password.

pm_addr

The IP address of the IPMI device.

mac

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

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.

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

After creating the template, run the following command to verify the formatting and syntax:

```
$ openstack baremetal instackenv validate --file instackenv.json
```

Save the file to the **stack** user's home directory (**/home/stack/instackenv.json**), then run the following command to import the template to the director:

```
$ openstack overcloud node import ~/instackenv.json
```

This imports the template and registers each node from the template into the director.

After the node registration and configuration completes, wiew a list of these nodes in the CLI:

```
$ openstack baremetal node list
```

5.2. INSPECTING THE HARDWARE OF NODES

The director can run an introspection process on each node. This process causes each node to boot an introspection agent over PXE. This agent collects hardware data from the node and sends it back to the director. The director then stores this introspection data in the OpenStack Object Storage (swift) service running on the director. The director uses hardware information for various purposes such as profile tagging, benchmarking, and manual root disk assignment.

> **NOTE**
>
> You can also create policy files to automatically tag nodes into profiles immediately after introspection. For more information on creating policy files and including them in the introspection process, see Appendix E, *Automatic Profile Tagging*. Alternatively, you can manually tag nodes into profiles as per the instructions in Section 5.3, “Tagging Nodes into Profiles”.

Set all nodes to a managed state:

```
$ for node in $(openstack baremetal node list -c UUID -f value) ; do openstack baremetal node
manage $node ; done
```

Run the following command to inspect the hardware attributes of each node:

```
$ openstack overcloud node introspect --all-manageable --provide
```

- The **--all-manageable** option introspects only nodes in a managed state. In this example, it is all of them.
- The **--provide** option resets all nodes to an active state after introspection.

Monitor the progress of the introspection using the following command in a separate terminal window:

```
$ sudo journalctl -l -u openstack-ironic-inspector -u openstack-ironic-inspector-dnsmasq -u
openstack-ironic-conductor -f
```

IMPORTANT

Make sure this process runs to completion. This process usually takes 15 minutes for bare metal nodes.

Alternatively, perform a single introspection on each node individually. Set the node to management mode, perform the introspection, then move the node out of management mode:

```
$ openstack baremetal node manage [NODE UUID]
$ openstack overcloud node introspect [NODE UUID] --provide
```

5.3. TAGGING NODES INTO PROFILES

After registering and inspecting the hardware of each node, you will tag them into specific profiles. These profile tags match your nodes to flavors, and in turn the flavors are assigned to a deployment role. The following example shows the relationship across roles, flavors, profiles, and nodes for Controller nodes:

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

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

NOTE

For a large number of nodes, use automatic profile tagging. See Appendix E, *Automatic Profile Tagging* for more details.

To tag a node into a specific profile, add a **profile** option to the **properties/capabilities** parameter for each node. For example, to tag your nodes to use Controller and Compute profiles respectively, use the following commands:

```
$ openstack baremetal node set --property capabilities='profile:compute,boot_option:local' 58c3d07e-24f2-48a7-bbb6-6843f0e8ee13
$ openstack baremetal node set --property capabilities='profile:control,boot_option:local' 1a4e30da-b6dc-499d-ba87-0bd8a3819bc0
```

The addition of the **profile:compute** and **profile:control** options tag the two nodes into each respective profiles.

These commands also set the **boot_option:local** parameter, which defines how each node boots. Depending on your hardware, you might also need to add the **boot_mode** parameter to **uefi** so that nodes boot using UEFI instead of the default BIOS mode. For more information, see Section D.2, “UEFI Boot Mode”.

After completing node tagging, check the assigned profiles or possible profiles:

```
$ openstack overcloud profiles list
```

Custom Role Profiles

If using custom roles, you might need to create additional flavors and profiles to accommodate these new roles. For example, to create a new flavor for a Networker role, run the following command:

```
$ openstack flavor create --id auto --ram 4096 --disk 40 --vcpus 1 networker
$ openstack flavor set --property "cpu_arch"="x86_64" --property "capabilities:boot_option"="local" --property "capabilities:profile"="networker" networker
```

Assign nodes with this new profile:

```
$ openstack baremetal node set --property capabilities='profile:networker,boot_option:local' dad05b82-0c74-40bf-9d12-193184bfc72d
```

5.4. DEFINING THE ROOT DISK FOR NODES

Some nodes might use multiple disks. This means the director needs to identify the disk to use for the root disk during provisioning. There are several properties you can use to help the 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 Only use this for devices with persistent names.

This example shows how to use a disk's serial number to specify the root disk to deploy the overcloud image.

Check the disk information from each node's hardware introspection. The following command displays the disk information from a node:

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

For example, the data for one node might show three disks:

```
[
  {
    "size": 299439751168,
    "rotational": true,
    "vendor": "DELL",
    "name": "/dev/sda",
    "wwn_vendor_extension": "0x1ea4dcc412a9632b",
    "wwn_with_extension": "0x61866da04f3807001ea4dcc412a9632b",
    "model": "PERC H330 Mini",
    "wwn": "0x61866da04f380700",
    "serial": "61866da04f3807001ea4dcc412a9632b"
  }
  {
    "size": 299439751168,
    "rotational": true,
    "vendor": "DELL",
    "name": "/dev/sdb",
    "wwn_vendor_extension": "0x1ea4e13c12e36ad6",
    "wwn_with_extension": "0x61866da04f380d001ea4e13c12e36ad6",
    "model": "PERC H330 Mini",
    "wwn": "0x61866da04f380d00",
    "serial": "61866da04f380d001ea4e13c12e36ad6"
  }
  {
    "size": 299439751168,
    "rotational": true,
    "vendor": "DELL",
    "name": "/dev/sdc",
    "wwn_vendor_extension": "0x1ea4e31e121cfb45",
    "wwn_with_extension": "0x61866da04f37fc001ea4e31e121cfb45",
    "model": "PERC H330 Mini",
    "wwn": "0x61866da04f37fc00",
    "serial": "61866da04f37fc001ea4e31e121cfb45"
  }
]
```

For this example, set the root device to disk 2, which has **61866da04f380d001ea4e13c12e36ad6** as the serial number. This requires a change to the **root_device** parameter for the node definition:

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

This helps the director identify the specific disk to use as the root disk. When we initiate our overcloud creation, the director provisions this node and writes the overcloud image to this disk.

NOTE

Make sure to configure the BIOS of each node to include booting from the chosen root disk. The recommended boot order is network boot, then root disk boot.

IMPORTANT

Do not use **name** to set the root disk as this value can change when the node boots.

5.5. CUSTOMIZING THE OVERCLOUD

The undercloud includes a set of Heat templates that acts as a plan for your overcloud creation. You can customize advanced features for your overcloud using the Advanced Overcloud Customization guide.

Otherwise, you can continue and deploy a basic overcloud. See Section 5.6, “Creating the Overcloud with the CLI Tools” for more information.

IMPORTANT

A basic overcloud uses local LVM storage for block storage, which is not a supported configuration. It is recommended to use an external storage solution for block storage.

5.6. CREATING THE OVERCLOUD WITH THE CLI TOOLS

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

WARNING

Do not run **openstack overcloud deploy** as a background process. The overcloud creation might hang in mid-deployment if started as a background process.

Setting Overcloud Parameters

The following table lists the additional parameters when using the **openstack overcloud deploy** command.

Table 5.2. Deployment Parameters

Parameter	Description
--templates [TEMPLATES]	The directory containing the Heat templates to deploy. If blank, the command uses the default template location at **/usr/share/openstack-tripleo-heat-templates/**
--stack STACK	The name of the stack to create or update
-t [TIMEOUT], --timeout [TIMEOUT]	Deployment timeout in minutes
--libvirt-type [LIBVIRT_TYPE]	Virtualization type to use for hypervisors
--ntp-server [NTP_SERVER]	Network Time Protocol (NTP) server 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 controllers are consistently referring to the same time source. Note that a typical environment might already have a designated NTP time source with established practices.
--no-proxy [NO_PROXY]	Defines custom values for the environment variable no_proxy, which excludes certain domain extensions from proxy communication.
--overcloud-ssh-user OVERCLOUD_SSH_USER	Defines the SSH user to access the overcloud nodes. Normally SSH access occurs through the **heat-admin** user.

Parameter	Description
-e [EXTRA HEAT TEMPLATE], **--extra-template [EXTRA HEAT TEMPLATE]**	Extra environment files to pass to the overcloud deployment. Can be specified more than once. Note that the order of environment files passed 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	The directory containing environment files to include in deployment. The command processes these environment files in numerical, then alphabetical order.
--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.
--dry-run	Performs validation check on the overcloud but does not actually create the overcloud.
--skip-postconfig	Skip the overcloud post-deployment configuration.
--force-postconfig	Force the overcloud post-deployment configuration.
--answers-file ANSWERS_FILE	Path to a YAML file with arguments and parameters.
--rhel-reg	Register overcloud nodes to the Customer Portal or Satellite 6.
--reg-method	Registration method to use for the overcloud nodes. **satellite** for Red Hat Satellite 6 or Red Hat Satellite 5, **portal** for Customer Portal.
--reg-org [REG_ORG]	Organization to use for registration.
--reg-force	Register the system even if it is already registered.

Parameter	Description
--reg-sat-url [REG_SAT_URL]	The base URL of the Satellite server to register overcloud nodes. Use the Satellite's 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 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 a Red Hat Satellite 5 server, the overcloud obtains the **RHN-ORG-TRUSTED-SSL-CERT** file and registers with **rhnreg_ks**.
--reg-activation-key [REG_ACTIVATION_KEY]	Activation key to use for registration.

Some command line parameters are outdated or deprecated in favor of using Heat template parameters, which you include in the **parameter_defaults** section on an environment file. The following table maps deprecated parameters to their Heat Template equivalents.

Table 5.3. 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 Cinder nodes to scale out	**BlockStorageCount**
--swift-storage-scale	The number of Swift nodes to scale out	**ObjectStorageCount**
--control-flavor	The flavor to use for Controller nodes	**OvercloudControlFlavor**
--compute-flavor	The flavor to use for Compute nodes	**OvercloudComputeFlavor**

Parameter	Description	Heat Template Parameter
--ceph-storage-flavor	The flavor to use for Ceph Storage nodes	**OvercloudCephStorageFlavor**
--block-storage-flavor	The flavor to use for Cinder nodes	**OvercloudBlockStorageFlavor**
--swift-storage-flavor	The flavor to use for Swift storage nodes	**OvercloudSwiftStorageFlavor**
--neutron-flat-networks	Defines the flat networks to configure in neutron plugins. Defaults to "datacentre" to permit external network creation	**NeutronFlatNetworks**
--neutron-physical-bridge	An Open vSwitch bridge to create on each hypervisor. This defaults to "br-ex". Typically, this should not need to be changed	**HypervisorNeutronPhysicalBridge**
--neutron-bridge-mappings	The logical to physical bridge mappings to use. Defaults to mapping the external bridge on hosts (br-ex) to a physical name (datacentre). You would use this for the default floating network	**NeutronBridgeMappings**
--neutron-public-interface	Defines the interface to bridge onto br-ex for network nodes	**NeutronPublicInterface**
--neutron-network-type	The tenant network type for Neutron	**NeutronNetworkType**
--neutron-tunnel-types	The tunnel types for the Neutron tenant network. To specify multiple values, use a comma separated string	**NeutronTunnelTypes**
--neutron-tunnel-id-ranges	Ranges of GRE tunnel IDs to make available for tenant network allocation	**NeutronTunnelIdRanges**
--neutron-vni-ranges	Ranges of VXLAN VNI IDs to make available for tenant network allocation	**NeutronVniRanges**

Parameter	Description	Heat Template Parameter
--neutron-network-vlan-ranges	The Neutron ML2 and Open vSwitch VLAN mapping range to support. Defaults to permitting any VLAN on the *datacentre* physical network	**NeutronNetworkVLANRanges**
--neutron-mechanism-drivers	The mechanism drivers for the neutron tenant network. Defaults to "openvswitch". To specify multiple values, use a comma-separated string	**NeutronMechanismDrivers**
--neutron-disable-tunneling	Disables tunneling in case you aim to use a VLAN segmented network or flat network with Neutron	No parameter mapping.
--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 as any errors can cause your deployment to fail.	No parameter mapping

These parameters are scheduled for removal in a future version of Red Hat OpenStack Platform.

NOTE

Run the following command for a full list of options:

```
$ openstack help overcloud deploy
```

5.7. INCLUDING ENVIRONMENT FILES IN OVERCLOUD CREATION

The **-e** includes an environment file to customize your overcloud. You can include as many environment files as necessary. However, the order of the environment files is important as the parameters and resources defined in subsequent environment files take precedence. Use the following list as an example of the environment file order:

- The amount of nodes per each role and their flavors. It is vital to include this information for overcloud creation.
- Any network isolation files, starting with the initialization file (**environments/network-isolation.yaml**) from the heat template collection, then your custom NIC configuration file, and finally any additional network configurations.
- Any external load balancing environment files.

- Any storage environment files such as Ceph Storage, NFS, iSCSI, etc.
- Any environment files for Red Hat CDN or Satellite registration.
- Any other custom environment files.

Any environment files added to the overcloud using the **-e** option become part of your overcloud's stack definition. The following command is an example of how to start the overcloud creation with custom environment files included:

```
$ openstack overcloud deploy --templates \
  -e ~/templates/node-info.yaml\
  -e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml \
  -e ~/templates/network-environment.yaml \
  -e ~/templates/storage-environment.yaml \
  --ntp-server pool.ntp.org \
```

This command contains the following additional options:

- **--templates** - Creates the overcloud using the Heat template collection in **/usr/share/openstack-tripleo-heat-templates**.
- **-e ~/templates/node-info.yaml** - The **-e** option adds an additional environment file to the overcloud deployment. In this case, it is a custom environment file that defines how many nodes and which flavors to use for each role. For example:

  ```
  parameter_defaults:
    OvercloudControlFlavor: control
    OvercloudComputeFlavor: compute
    OvercloudCephStorageFlavor: ceph-storage
    ControllerCount: 3
    ComputeCount: 3
    CephStorageCount: 3
  ```

- **-e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml** - The **-e** option adds an additional environment file to the overcloud deployment. In this case, it is an environment file that initializes network isolation configuration.
- **-e ~/templates/network-environment.yaml** - The **-e** option adds an additional environment file to the overcloud deployment. In this case, it is a network environment file that contains customization for network isolation.
- **-e ~/templates/storage-environment.yaml** - The **-e** option adds an additional environment file to the overcloud deployment. In this case, it is a custom environment file that initializes our storage configuration.
- **--ntp-server pool.ntp.org** - Use an NTP server for time synchronization. This is useful for keeping the Controller node cluster in synchronization.

The director requires these environment files for re-deployment and post-deployment functions in Chapter 7, *Performing Tasks after Overcloud Creation*. Failure to include these files can result in damage to your overcloud.

If you aim to later modify the overcloud configuration, you should:

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 as such manual configuration gets overridden by the director's configuration when updating the overcloud stack with the director.

Including an Environment File Directory

You can add a whole directory containing environment files using the **--environment-directory** option. The deployment command processes the environment files in this directory in numerical, then alphabetical order. If using this method, it is recommended to use filenames with a numerical prefix to order how they are processed. For example:

```
$ ls -1 ~/templates
00-node-info.yaml
10-network-isolation.yaml
20-network-environment.yaml
30-storage-environment.yaml
40-rhel-registration.yaml
```

Run the following deployment command to include the directory:

```
$ openstack overcloud deploy --templates --environment-directory ~/templates
```

Using an Answers File

An answers file is a YAML format file that simplifies the inclusion of templates and environment files. The answers file uses the following parameters:

templates

The core Heat template collection to use. This acts as a substitute for the **--templates** command line option.

environments

A list of environment files to include. This acts as a substitute for the **--environment-file** (**-e**) command line option.

For example, an answers file might contain the following:

```
templates: /usr/share/openstack-tripleo-heat-templates/
environments:
  - ~/templates/00-node-info.yaml
  - ~/templates/10-network-isolation.yaml
  - ~/templates/20-network-environment.yaml
  - ~/templates/30-storage-environment.yaml
  - ~/templates/40-rhel-registration.yaml
```

Run the following deployment command to include the answers file:

```
$ openstack overcloud deploy --answers-file ~/answers.yaml
```

5.8. MANAGING OVERCLOUD PLANS

As an alternative to using the **openstack overcloud deploy** command, the director can also manage imported plans.

To create a new plan, run the following command as the **stack** user:

```
$ openstack overcloud plan create --templates /usr/share/openstack-tripleo-heat-templates my-overcloud
```

This creates a plan from the core Heat template collection in **/usr/share/openstack-tripleo-heat-templates**. The director names the plan based on your input. In this example, it is **my-overcloud**. The director uses this name as a label for the object storage container, the workflow environment, and overcloud stack names.

Add parameters from environment files using the following command:

```
$ openstack overcloud parameters set my-overcloud ~/templates/my-environment.yaml
```

Deploy your plans using the following command:

```
$ openstack overcloud plan deploy my-overcloud
```

Delete existing plans using the following command:

```
$ openstack overcloud plan delete my-overcloud
```

NOTE

The **openstack overcloud deploy** command essentially uses all of these commands to remove the existing plan, upload a new plan with environment files, and deploy the plan.

5.9. VALIDATING OVERCLOUD TEMPLATES AND PLANS

Before executing an overcloud creation or stack update, validate your Heat templates and environment files for any errors.

Creating a Rendered Template

The core Heat templates for the overcloud are in a Jinja2 format. To validate your templates, render a version without Jinja2 formatting using the following commands:

```
$ openstack overcloud plan create --templates /usr/share/openstack-tripleo-heat-templates overcloud-validation
$ mkdir ~/overcloud-validation
$ cd ~/overcloud-validation
$ swift download overcloud-validation
```

Use the rendered template in **~/overcloud-validation** for the validation tests that follow.

Validating Template Syntax

Use the following command to validate the template syntax:

```
$ openstack orchestration template validate --show-nested --template ~/overcloud-validation/overcloud.yaml -e ~/overcloud-validation/overcloud-resource-registry-puppet.yaml -e [ENVIRONMENT FILE] -e [ENVIRONMENT FILE]
```

NOTE

The validation requires the **overcloud-resource-registry-puppet.yaml** environment file to include overcloud-specific resources. Add any additional environment files to this command with **-e** option. Also include the **--show-nested** option to resolve parameters from nested templates.

This command identifies any syntax errors in the template. If the template syntax validates successfully, the output shows a preview of the resulting overcloud template.

5.10. MONITORING THE OVERCLOUD CREATION

The overcloud creation process begins and the director provisions your nodes. This process takes some time to complete. To view the status of the overcloud creation, open a separate terminal as the **stack** user and run:

```
$ source ~/stackrc
$ openstack stack list --nested
```

The **openstack stack list --nested** command shows the current stage of the overcloud creation.

5.11. ACCESSING THE OVERCLOUD

The director generates a script to configure and help authenticate interactions with your overcloud from the director host. The director saves this file, **overcloudrc**, in your **stack** user's home director. Run the following command to use this file:

```
$ source ~/overcloudrc
```

This loads the necessary environment variables to interact with your overcloud from the director host's CLI. To return to interacting with the director's host, run the following command:

```
$ source ~/stackrc
```

Each node in the overcloud also contains a user called **heat-admin**. The **stack** user has SSH access to this user on each node. To access a node over SSH, find the IP address of the desired node:

```
$ nova list
```

Then connect to the node using the **heat-admin** user and the node's IP address:

```
$ ssh heat-admin@192.0.2.23
```

5.12. COMPLETING THE OVERCLOUD CREATION

This concludes the creation of the overcloud using the command line tools. For post-creation functions, see Chapter 7, *Performing Tasks after Overcloud Creation*.

CHAPTER 6. CONFIGURING BASIC OVERCLOUD REQUIREMENTS WITH THE WEB UI

This chapter provides the basic configuration steps for an OpenStack Platform environment using the web UI. An overcloud with a basic configuration contains no custom features. However, you can add advanced configuration options to this basic overcloud and customize it to your specifications using the instructions in the Advanced Overcloud Customization guide.

For the examples in this chapter, all nodes in this chapter are bare metal systems using IPMI for power management. For more supported power management types and their options, see Appendix B, *Power Management Drivers*.

Workflow

1. Register blank nodes using a node definition template and manual registration.
2. Inspect hardware of all nodes.
3. Upload an overcloud plan to the director.
4. Assign nodes into roles.

Requirements

- The director node created in Chapter 4, *Installing the Undercloud* with the UI enabled
- A set of bare metal machines for your nodes. The number of node required depends on the type of overcloud you intend to create (see Section 3.1, "Planning Node Deployment Roles" for information on overcloud roles). These machines also must comply with the requirements set for each node type. For these requirements, see Section 2.4, "Overcloud Requirements". These nodes do not require an operating system. The director copies a Red Hat Enterprise Linux 7 image to each node.
- One network connection for our Provisioning network, which is configured as a native VLAN. All nodes must connect to this network and comply with the requirements set in Section 2.3, "Networking Requirements".
- All other network types use the Provisioning network for OpenStack services. However, you can create additional networks for other network traffic types.

6.1. ACCESSING THE WEB UI

The UI runs as a standalone application in your browser. However, your client system requires access to the Public API endpoints for the following components on the undercloud:

Component	UI Purpose
OpenStack Identity (**keystone**)	For authentication to the UI and for endpoint discovery of other services.
OpenStack Orchestration (**heat**)	For the status of the deployment.

Component	UI Purpose
OpenStack Bare Metal (**ironic**)	For control of nodes.
OpenStack Object Storage (**swift**)	For storage of the Heat template collection or plan used for the overcloud creation.
OpenStack Workflow (**mistral**)	To access and execute director tasks.
OpenStack Messaging (**zaqar**)	A websocket-based service to find the status of certain tasks.

The UI interacts directly with these Public APIs, which is why your client system requires access to their endpoints.

IMPORTANT

Accessing the director's web UI using Mozilla Firefox will require certain server identity exceptions for these OpenStack Platform Public APIs. See Appendix F, *Firefox Server Exceptions for UI Access* for information on implementing these exceptions.

Users access the director's web UI through SSL. For example, if the IP address of your undercloud is 192.0.2.1, then the address to access the UI is **https://192.0.2.1**. The web UI initially presents a login screen with fields for the following:

- Username - The administration user for the director. The default is **admin**.
- Password - The password for the administration user. Run **sudo hiera admin_password** as the **stack** user on the undercloud host terminal to find out the password.

When logging in to the UI, the UI accesses the OpenStack Identity Public API and obtains the endpoints for the other Public API services. However, if you aim to change the endpoint or use a different IP for endpoint access, the director UI reads settings from the **/var/www/openstack-tripleo-ui/dist/tripleo_ui_config.js** file. This file uses the following parameters:

Parameter	Description
keystone	The Public API for the OpenStack Identity (**keystone**) service. The UI automatically discovers the endpoints for the other services through this service, which means you only need to define this parameter. However, you can define custom URLs for the other endpoints if necessary.
heat	The Public API for the OpenStack Orchestration (**heat**) service. Requires the URL in the format: **https://<IP>:13004//v1/<tenant_id>** where **tenant_id** is the **admin** tenant of the undercloud.

Parameter	Description
ironic	The Public API for the OpenStack Bare Metal (**ironic**) service.
swift	The Public API for the OpenStack Object Storage (**swift**) service. Requires the URL in the format: **https://<IP>:8080/v1/AUTH_<tenant_id>** where **tenant_id** is the **admin** tenant of the undercloud.
mistral	The Public API for the OpenStack Workflow (**mistral**) service.
zaqar-websocket	The websocket for the OpenStack Messaging (**zaqar**) service.
zaqar_default_queue	The messaging queue to use for the OpenStack Messaging (**zaqar**) service. The default is **tripleo**.

The following is an example **tripleo_ui_config.js** file:

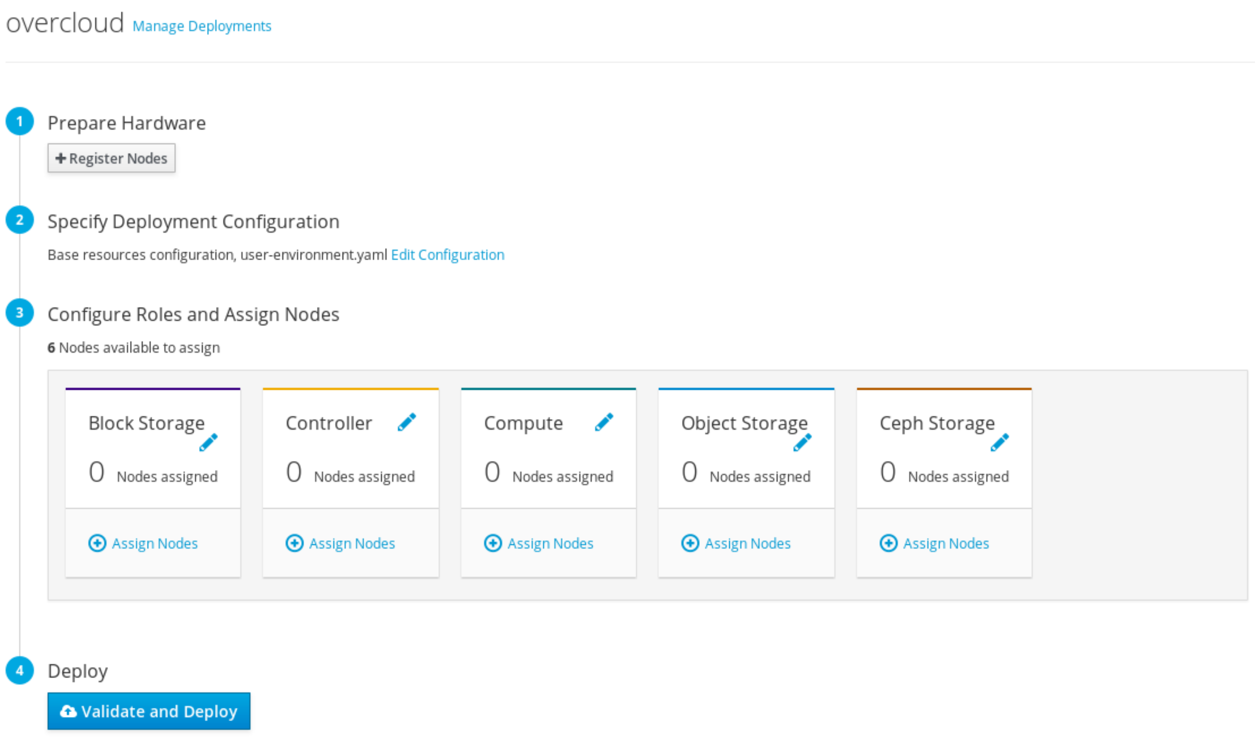
```
window.tripleOUiConfig = {
  "keystone": "https://192.0.2.2:13000/v2.0",
  "heat": "https://192.0.2.2:13004/v1/f30d815864be456dbc2a00f4e9200e9e",
  "ironic": "https://192.0.2.2:13385",
  "mistral": "https://192.0.2.2:13989/v2",
  "swift": "https://192.0.2.2:13808/v1/AUTH_f30d815864be456dbc2a00f4e9200e9e",
  "zaqar-websocket": "wss://192.0.2.2:9000",
  "zaqar_default_queue": "tripleo"
};
```

6.2. NAVIGATING THE WEB UI

The UI provides three main sections:

Deployment Plan

A menu item at the top of the UI. This page acts as the main UI section and allows you to define the plan to use for your overcloud creation, the nodes to assign to each role, and the status of the current overcloud. This section also provides a deployment workflow to guide you through each step of the overcloud creation process, including setting deployment parameters and assigning your nodes to roles.



overcloud Manage Deployments

1 Prepare Hardware

2 Specify Deployment Configuration

Base resources configuration, user-environment.yaml Edit Configuration

3 Configure Roles and Assign Nodes

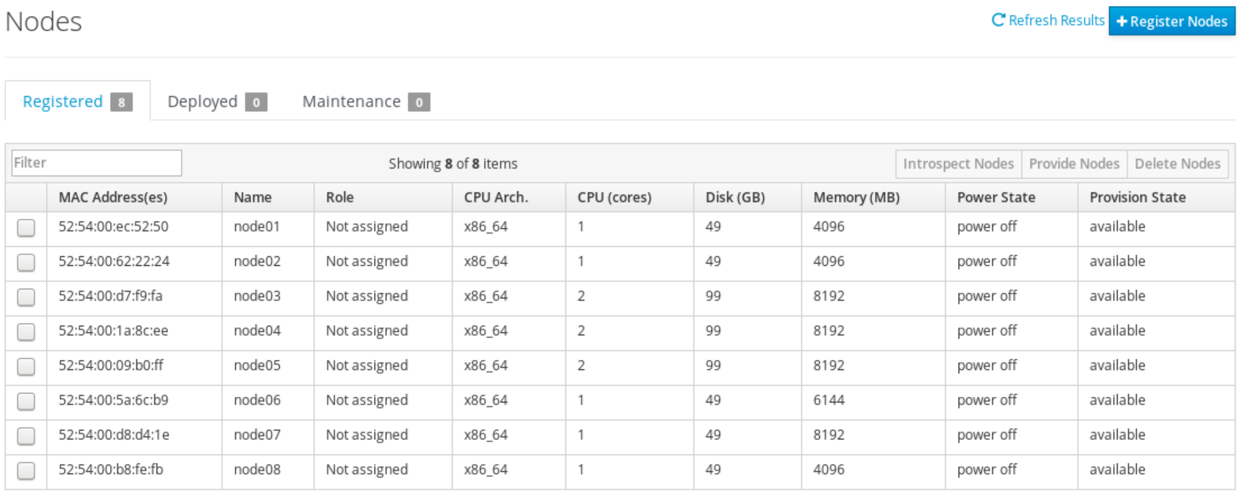
6 Nodes available to assign

Block Storage	Controller	Compute	Object Storage	Ceph Storage
0 Nodes assigned	0 Nodes assigned	0 Nodes assigned	0 Nodes assigned	0 Nodes assigned
Assign Nodes	Assign Nodes	Assign Nodes	Assign Nodes	Assign Nodes

4 Deploy

Nodes

A menu item at the top of the UI. This page acts as a node configuration section and provides methods for registering new nodes and introspecting registered nodes. This section also defines the nodes available for deployment, the nodes currently deployed, and nodes in maintenance.



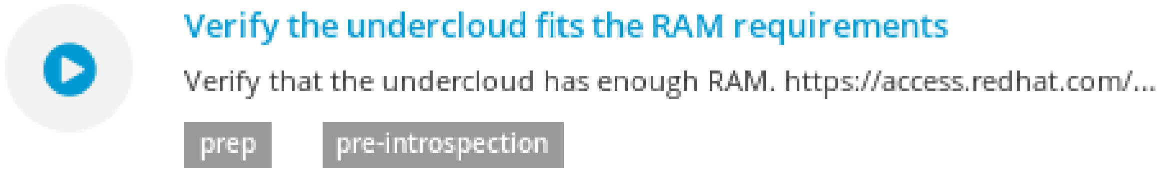
Nodes

Showing **8** of **8** items

	MAC Address(es)	Name	Role	CPU Arch.	CPU (cores)	Disk (GB)	Memory (MB)	Power State	Provision State
□	52:54:00:ec:52:50	node01	Not assigned	x86_64	1	49	4096	power off	available
□	52:54:00:62:22:24	node02	Not assigned	x86_64	1	49	4096	power off	available
□	52:54:00:d7:f9:fa	node03	Not assigned	x86_64	2	99	8192	power off	available
□	52:54:00:1a:8c:ee	node04	Not assigned	x86_64	2	99	8192	power off	available
□	52:54:00:09:b0:ff	node05	Not assigned	x86_64	2	99	8192	power off	available
□	52:54:00:5a:6c:b9	node06	Not assigned	x86_64	1	49	6144	power off	available
□	52:54:00:d8:d4:1e	node07	Not assigned	x86_64	1	49	8192	power off	available
□	52:54:00:b8:fe:fb	node08	Not assigned	x86_64	1	49	4096	power off	available

Validations

A side panel on the right side of the page. This section provides a set of pre-deployment and post-deployment system checks to ensure a successful overcloud creation process. These validation tasks run automatically at certain points in the deployment. However, you can also run them manually. Click the **Play** button for a validation task you want to run. Click the title of each validation task to run it, or click a validation title to view more information about it.



Verify the undercloud fits the RAM requirements

Verify that the undercloud has enough RAM. https://access.redhat.com/...

6.3. IMPORTING AN OVERCLOUD PLAN IN THE WEB UI

The director UI requires a plan before configuring the overcloud. This plan is usually a Heat template collection, like the one on your undercloud at **/usr/share/openstack-tripleo-heat-templates**. In addition, you can customize the plan to suit your hardware and environment requirements. For more information about customizing the overcloud, see the Advanced Overcloud Customization guide.

The plan displays four main steps to configuring your overcloud:

1. **Prepare Hardware** - Node registration and introspection.

2. **Specify Deployment Configuration** - Configuring overcloud parameters and defining the environment files to include.

3. **Configure Roles and Assign Nodes**- Assign nodes to roles and modify role-specific parameters.

4. **Deploy** - Launch the creation of your overcloud.

The undercloud installation and configuration automatically uploads a plan. You can also import multiple plans in the web UI. Click on **Manage Deployments** on the **Deployment Plan** screen. This displays the current **Plans** table.

Click **Create New Plan** and a window appears asking you for the following information:

- **Plan Name** - A plain text name for the plan. For example **overcloud**.

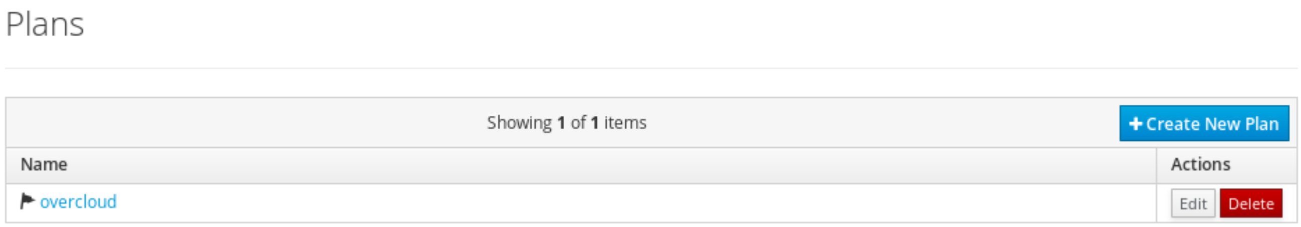
- **Upload Type** - Choose whether to upload a **Tar Archive (tar.gz)** or a full **Local Folder** (Google Chrome only).

- **Plan Files** - Click browser to choose the plan on your local file system.

If you need to copy the director's Heat template collection to a client machine, archive the files and copy them:

```
$ cd /usr/share/openstack-tripleo-heat-templates/
$ tar -cf ~/overcloud.tar *
$ scp ~/overcloud.tar user@10.0.0.55:~/.
```

Once the director UI uploads the plan, the plan appears in the **Plans** table and you can now configure it. Click on the **Deployment Plan**.



Plans

Showing 1 of 1 items

Name	Actions
overcloud	Edit Delete

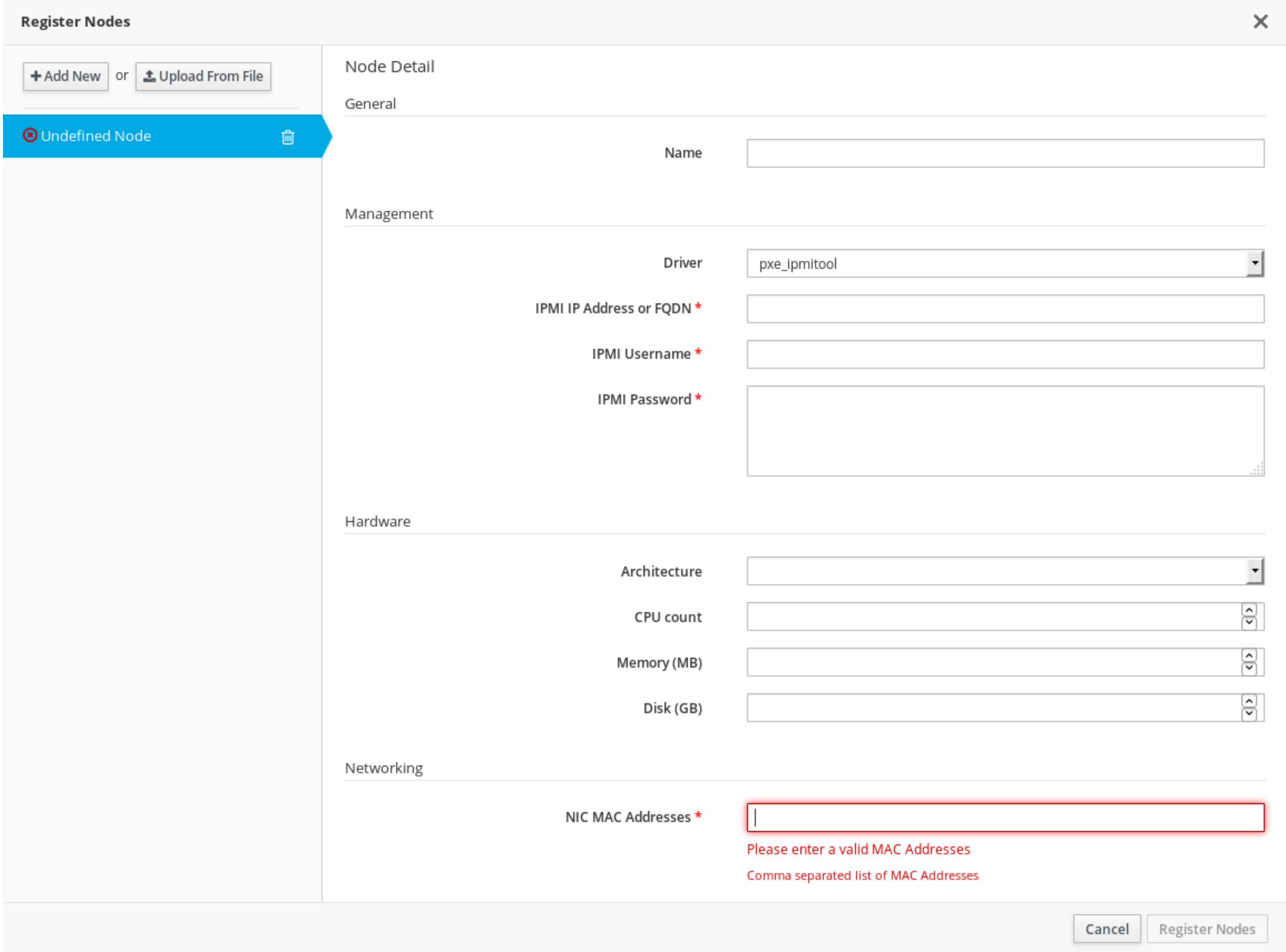
6.4. REGISTERING NODES IN THE WEB UI

The first step in configuring the overcloud is to register your nodes. Start the node registration process either through:

- Clicking **Register Nodes** under **1 Prepare Hardware** on the **Deployment Plan** screen.

- Clicking **Register Nodes** on the **Nodes** screen.

This displays the **Register Nodes** window.



Driver: pxe_ipmitool

Please enter a valid MAC Addresses

Comma separated list of MAC Addresses

The director requires a list of nodes for registration, which you can supply using one of two methods:

1. **Uploading a node definition template** - This involves clicking the **Upload from File** button and selecting a file. See Section 5.1, "Registering Nodes for the Overcloud" for the syntax of the node definition template.

2. **Manually registering each node**- This involves clicking **Add New** and providing a set of details for the node.

The details you need to provide for manual registration include the following:

Name

A plain text name for the node. Use only *RFC3986* unreserved characters.

Driver

The power management driver to use. This example uses the IPMI driver (**pxe_ipmitool**).

IPMI IP Address

The IP address of the IPMI device.

IPMI Username; IPMI Password

The IPMI username and password.

Architecture

(Optional) The system architecture.

CPU count

(Optional) The number of CPUs on the node.

Memory (MB)

(Optional) The amount of memory in MB.

Disk (GB)

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

NIC MAC Addresses

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

> NOTE
>
> The UI also allows for registration of nodes from a KVM host using the **pxe_ssh** driver. Note that this option is available for testing and evaluation purposes only. It is not recommended for Red Hat OpenStack Platform enterprise environments. For more information, see Section B.5, “SSH and Virsh” .

After entering your node information, click **Register Nodes** at the bottom of the window.

The director registers the nodes. Once complete, you can use the UI to perform introspection on the nodes.

6.5. INSPECTING THE HARDWARE OF NODES IN THE WEB UI

The director UI can run an introspection process on each node. This process causes each node to boot an introspection agent over PXE. This agent collects hardware data from the node and sends it back to the director. The director then stores this introspection data in the OpenStack Object Storage (swift) service running on the director. The director uses hardware information for various purposes such as profile tagging, benchmarking, and manual root disk assignment.

> NOTE
>
> You can also create policy files to automatically tag nodes into profiles immediately after introspection. For more information on creating policy files and including them in the introspection process, see Appendix E, *Automatic Profile Tagging*. Alternatively, you can tag nodes into profiles through the UI. See Section 6.7, “Tagging Nodes into Roles in the Web UI” for details on manually tagging nodes.

To start the introspection process:

1. Navigate to the **Nodes** screen

2. Select all nodes you aim to introspect.

3. Click **Introspect Nodes**

> IMPORTANT
>
> Make sure this process runs to completion. This process usually takes 15 minutes for bare metal nodes.

Once the introspection process completes, select all nodes with the **Provision State** set to **manageable** then click the **Provide Nodes** button. Wait until the **Provision State** changes to **available**.

The nodes are now ready to provision.

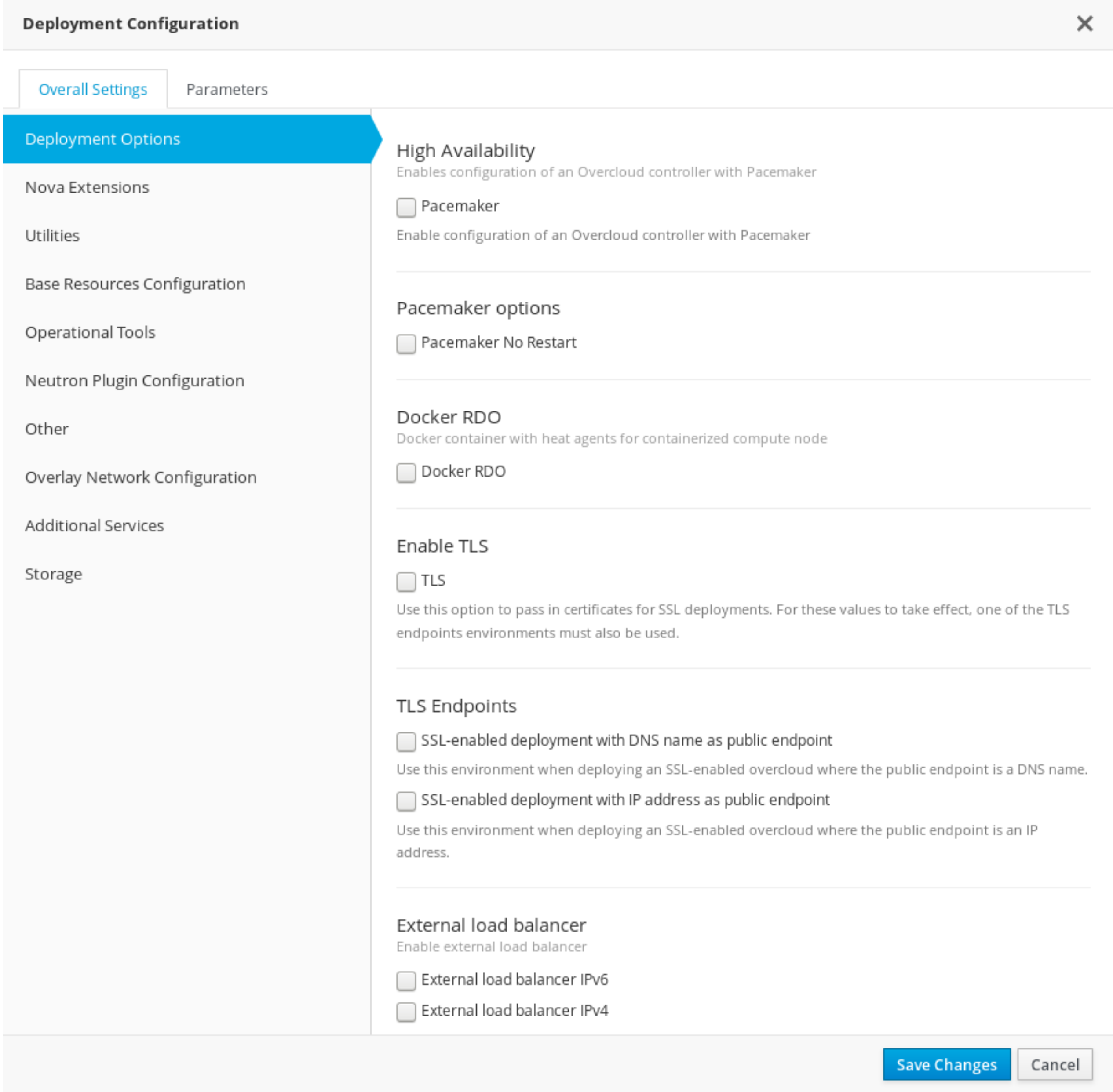
6.6. EDITING OVERCLOUD PLAN PARAMETERS IN THE WEB UI

The **Deployment Plan** screen provides a method to customize your uploaded plan. Under **2 Specify Deployment Configuration**, click the **Edit Configuration** link to modify your base overcloud configuration.

A window appears with two main tabs:

Overall Settings

This provides a method to include different features from your overcloud. These features are defined in the plan's **capabilities-map.yaml** file, which each feature using a different environment file. For example, under **Storage** you can select **Storage Environment**, which the plan maps to the **environments/storage-environment.yaml** file and allows you to configure NFS, iSCSI, or Ceph settings for your overcloud. The **Other** tab contains any environment files detected in the plan but not listed in the **capabilities-map.yaml**, which is useful for adding custom environment files included in the plan. Once you have selected the features to include, click **Save**.



High Availability

Enables configuration of an Overcloud controller with Pacemaker

□ Pacemaker

Enable configuration of an Overcloud controller with Pacemaker

Pacemaker options

□ Pacemaker No Restart

Docker RDO

Docker container with heat agents for containerized compute node

□ Docker RDO

Enable TLS

□ TLS

Use this option to pass in certificates for SSL deployments. For these values to take effect, one of the TLS endpoints environments must also be used.

TLS Endpoints

□ SSL-enabled deployment with DNS name as public endpoint

Use this environment when deploying an SSL-enabled overcloud where the public endpoint is a DNS name.

□ SSL-enabled deployment with IP address as public endpoint

Use this environment when deploying an SSL-enabled overcloud where the public endpoint is an IP address.

External load balancer

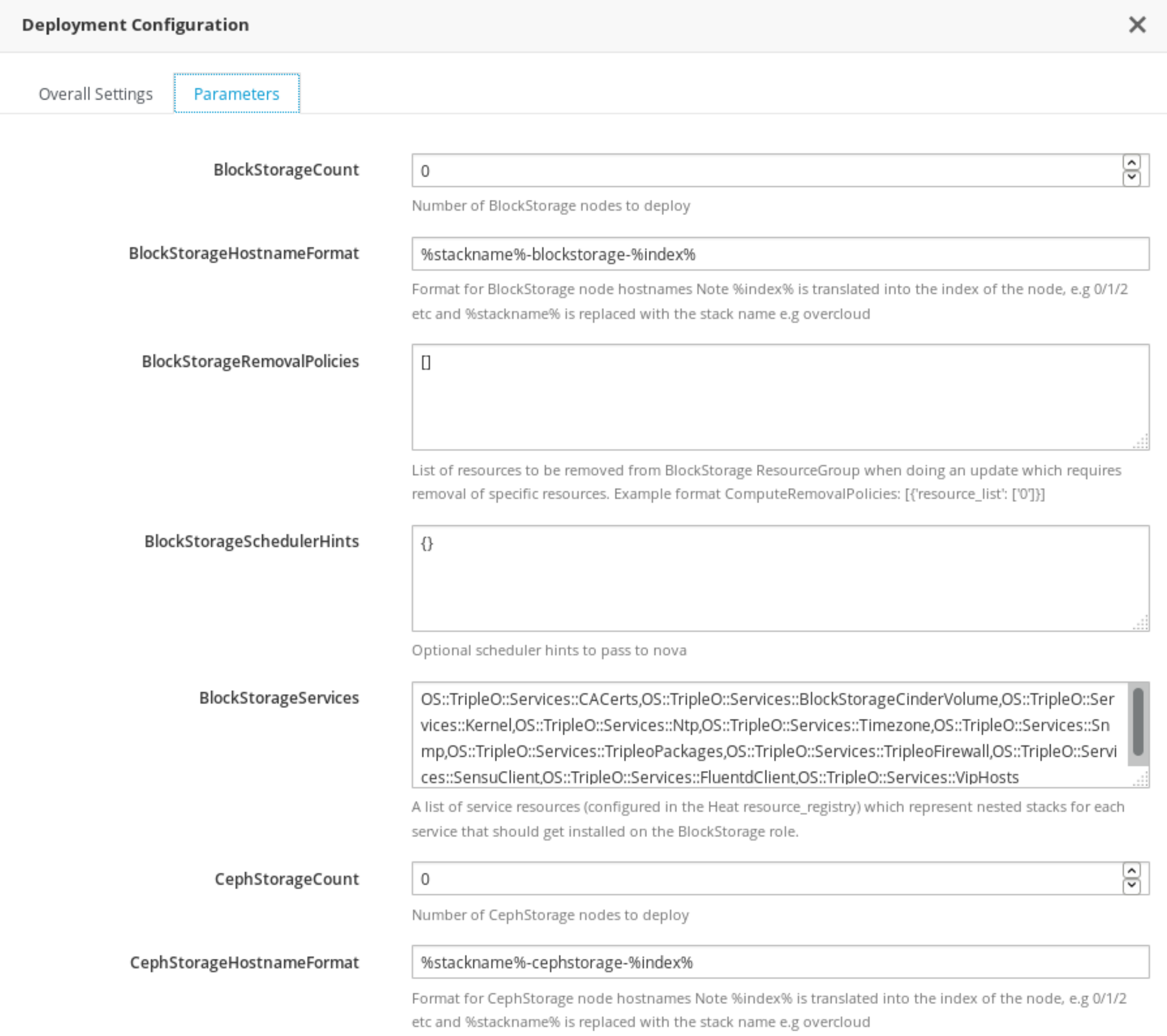
Enable external load balancer

□ External load balancer IPv6

□ External load balancer IPv4

Parameters

This includes various base-level and environment file parameters for your overcloud. For example, you can change the node count for each role in this section. If you aim to use three controller nodes, change the **ControllerCount** to 3. Once you have modified your base-level parameters, click **Save**.



BlockStorageCount: 0

Number of BlockStorage nodes to deploy

BlockStorageHostnameFormat: %stackname%-blockstorage-%index%

Format for BlockStorage node hostnames Note %index% is translated into the index of the node, e.g 0/1/2 etc and %stackname% is replaced with the stack name e.g overcloud

BlockStorageRemovalPolicies: []

List of resources to be removed from BlockStorage ResourceGroup when doing an update which requires removal of specific resources. Example format ComputeRemovalPolicies: [{'resource_list': ['0']}]

BlockStorageSchedulerHints: {}

Optional scheduler hints to pass to nova

BlockStorageServices: OS::TripleO::Services::CACerts,OS::TripleO::Services::BlockStorageCinderVolume,OS::TripleO::Services::Kernel,OS::TripleO::Services::Ntp,OS::TripleO::Services::Timezone,OS::TripleO::Services::Snmp,OS::TripleO::Services::TripleoPackages,OS::TripleO::Services::TripleoFirewall,OS::TripleO::Services::SensuClient,OS::TripleO::Services::FluentdClient,OS::TripleO::Services::VipHosts

A list of service resources (configured in the Heat resource_registry) which represent nested stacks for each service that should get installed on the BlockStorage role.

CephStorageCount: 0

Number of CephStorage nodes to deploy

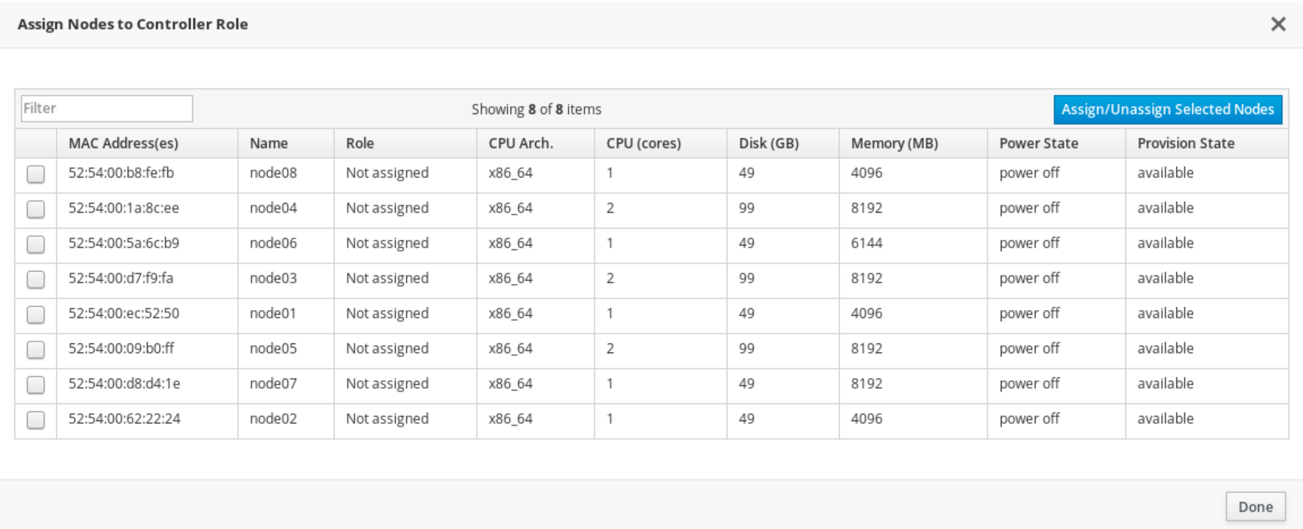
CephStorageHostnameFormat: %stackname%-cephstorage-%index%

Format for CephStorage node hostnames Note %index% is translated into the index of the node, e.g 0/1/2 etc and %stackname% is replaced with the stack name e.g overcloud

6.7. TAGGING NODES INTO ROLES IN THE WEB UI

After registering and inspecting the hardware of each node, you tag them into specific profiles. These profiles match your nodes to a specific flavor and deployment role.

To assign nodes to a role, scroll to the **3 Configure Roles and Assign Nodes**section on the **Deployment Plan** screen. Click **Assign Nodes** for a chosen role. A window appears that allows you to select the nodes to assign to the role. Once you have selected the role’s nodes, click **Assign/Unassign Selected Nodes**.


Assign Nodes to Controller Role
✕
Filter
Assign/Unassign Selected Nodes
Done

Showing **8** of **8** items

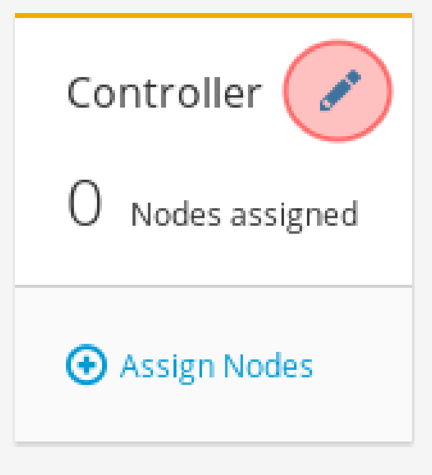
	MAC Address(es)	Name	Role	CPU Arch.	CPU (cores)	Disk (GB)	Memory (MB)	Power State	Provision State
□	52:54:00:b8:fe:fb	node08	Not assigned	x86_64	1	49	4096	power off	available
□	52:54:00:1a:8c:ee	node04	Not assigned	x86_64	2	99	8192	power off	available
□	52:54:00:5a:6c:b9	node06	Not assigned	x86_64	1	49	6144	power off	available
□	52:54:00:d7:f9:fa	node03	Not assigned	x86_64	2	99	8192	power off	available
□	52:54:00:ec:52:50	node01	Not assigned	x86_64	1	49	4096	power off	available
□	52:54:00:09:b0:ff	node05	Not assigned	x86_64	2	99	8192	power off	available
□	52:54:00:d8:d4:1e	node07	Not assigned	x86_64	1	49	8192	power off	available
□	52:54:00:62:22:24	node02	Not assigned	x86_64	1	49	4096	power off	available

Once these nodes are assigned to the role, click **Done** to return to the **Deployment Plan** screen.

Complete this task for each role you want in your overcloud.

6.8. EDITING NODES IN THE WEB UI

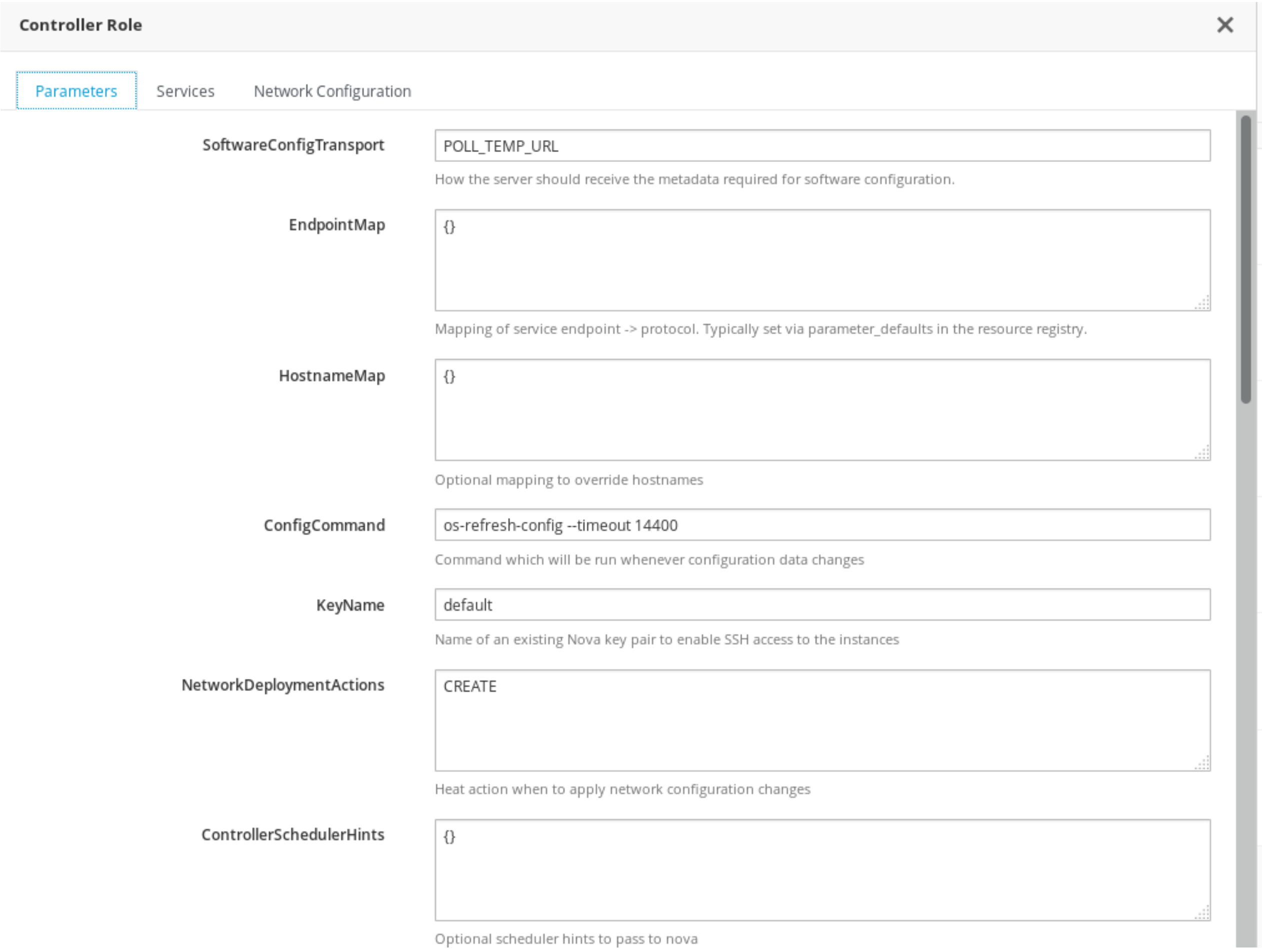
Each node role provides a method for configuring role-specific parameters. Scroll to **3 Configure Roles and Assign Nodes** roles on the **Deployment Plan** screen. Click the **Edit Role Parameters** icon (pencil icon) next to the role name.



A window appears that shows two main tabs:

Parameters

This includes various role specific parameters. For example, if you are editing the controller role, you can change the default flavor for the role using the **OvercloudControlFlavor** parameter. Once you have modified your role specific parameters, click **Save Changes**.



SoftwareConfigTransport: POLL_TEMP_URL
How the server should receive the metadata required for software configuration.

EndpointMap: {}
Mapping of service endpoint -> protocol. Typically set via parameter_defaults in the resource registry.

HostnameMap: {}
Optional mapping to override hostnames

ConfigCommand: os-refresh-config --timeout 14400
Command which will be run whenever configuration data changes

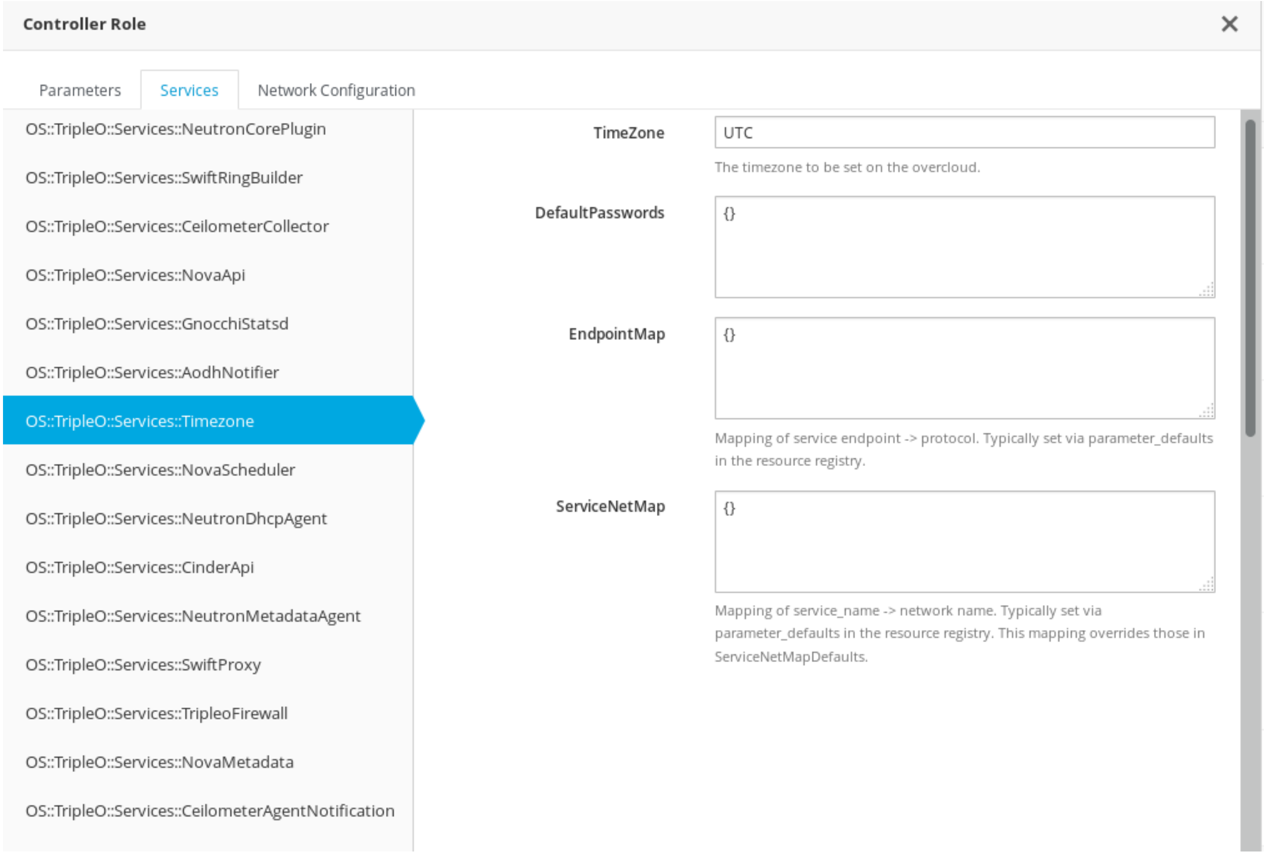
KeyName: default
Name of an existing Nova key pair to enable SSH access to the instances

NetworkDeploymentActions: CREATE
Heat action when to apply network configuration changes

ControllerSchedulerHints: {}
Optional scheduler hints to pass to nova

Services

This defines the service-specific parameters for the chosen role. The left panel shows a list of services that you select and modify. For example, to change the time zone, click the **OS::TripleO:Services:Timezone** service and change the **TimeZone** parameter to your desired time zone. Once you have modified your service-specific parameters, click **Save Changes**.



Controller Role

- OS::TripleO::Services::NeutronCorePlugin
- OS::TripleO::Services::SwiftRingBuilder
- OS::TripleO::Services::CeilometerCollector
- OS::TripleO::Services::NovaApi
- OS::TripleO::Services::GnocchiStatsd
- OS::TripleO::Services::AodhNotifier
- OS::TripleO::Services::Timezone
- OS::TripleO::Services::NovaScheduler
- OS::TripleO::Services::NeutronDhcpAgent
- OS::TripleO::Services::CinderApi
- OS::TripleO::Services::NeutronMetadataAgent
- OS::TripleO::Services::SwiftProxy
- OS::TripleO::Services::TripleoFirewall
- OS::TripleO::Services::NovaMetadata
- OS::TripleO::Services::CeilometerAgentNotification

TimeZone: UTC

The timezone to be set on the overcloud.

DefaultPasswords: {}

EndpointMap: {}

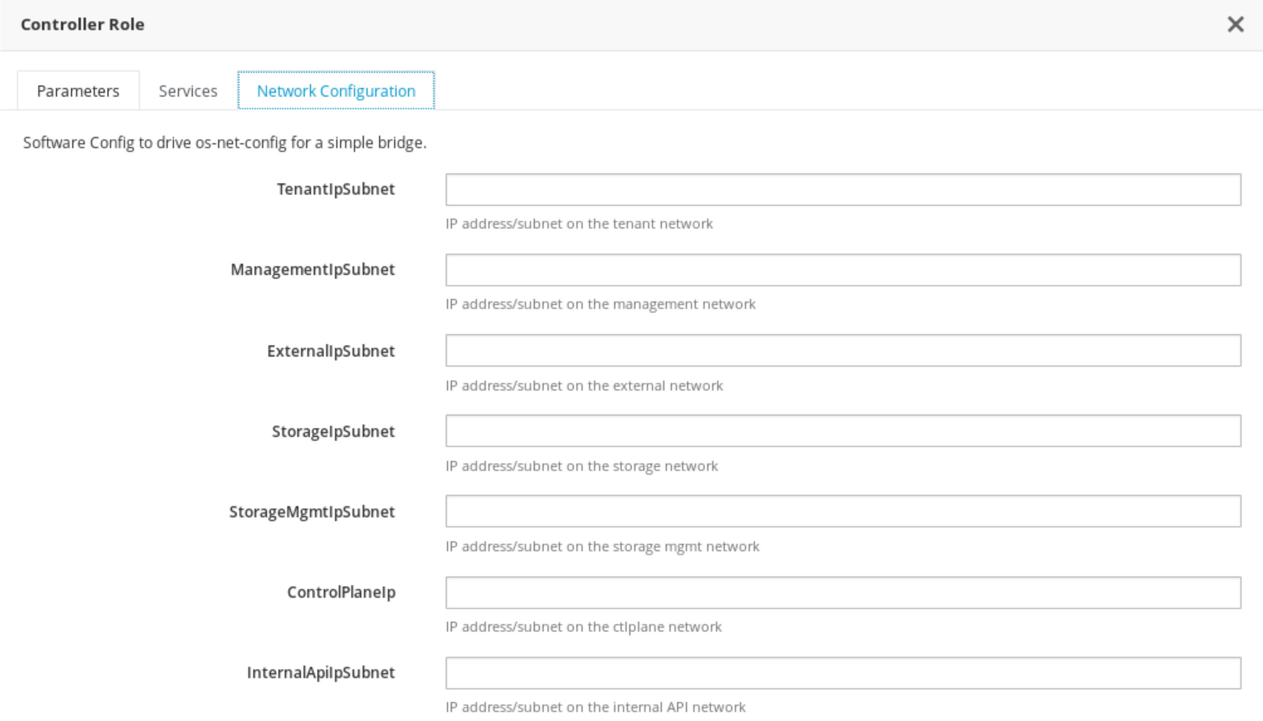
Mapping of service endpoint -> protocol. Typically set via parameter_defaults in the resource registry.

ServiceNetMap: {}

Mapping of service_name -> network name. Typically set via parameter_defaults in the resource registry. This mapping overrides those in ServiceNetMapDefaults.

Network Configuration

This allows you to define an IP address or subnet range for various networks in your overcloud.



Controller Role

Software Config to drive os-net-config for a simple bridge.

TenantIpSubnet

IP address/subnet on the tenant network

ManagementIpSubnet

IP address/subnet on the management network

ExternalIpSubnet

IP address/subnet on the external network

StorageIpSubnet

IP address/subnet on the storage network

StorageMgmtIpSubnet

IP address/subnet on the storage mgmt network

ControlPlaneIp

IP address/subnet on the ctlplane network

InternalApiIpSubnet

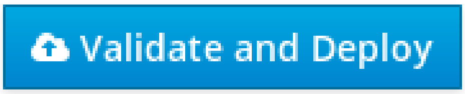
IP address/subnet on the internal API network

IMPORTANT

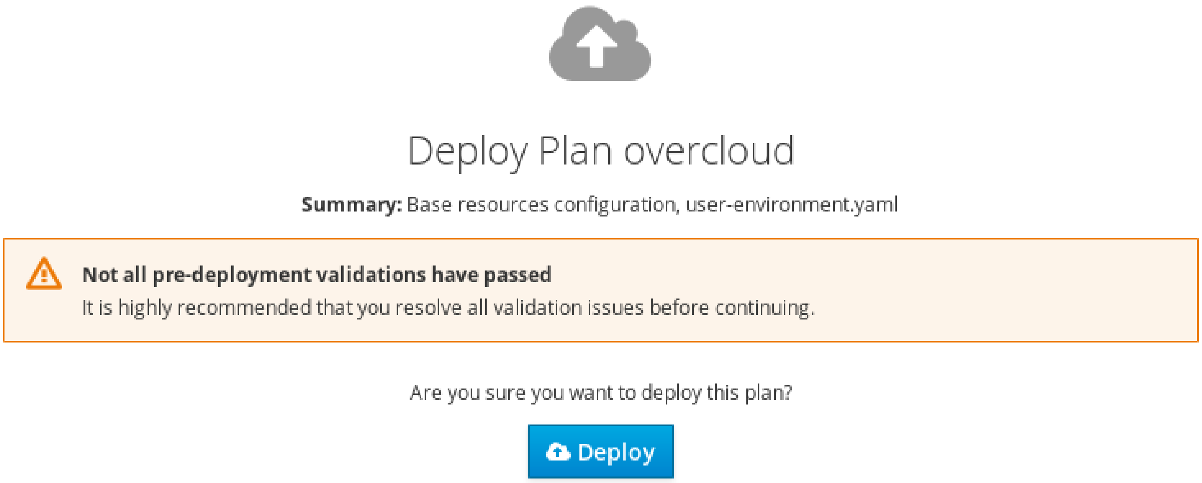
Although the role's service parameters appear in the UI, some services might be disabled by default. You can enable these services through the instructions in Section 6.6, "Editing Overcloud Plan Parameters in the Web UI". See also the *Composable Roles* section of the Advanced Overcloud Customization guide for information on enabling these services.

6.9. STARTING THE OVERCLOUD CREATION IN THE WEB UI

Once the overcloud plan is configured, you can start the overcloud deployment. This involves scrolling to the **4 Deploy** section and clicking **Validate and Deploy**.



If you have not run or passed all the validations for the undercloud, a warning message appears. Make sure that your undercloud host satisfies the requirements before running a deployment.



Deploy Plan overcloud

Summary: Base resources configuration, user-environment.yaml

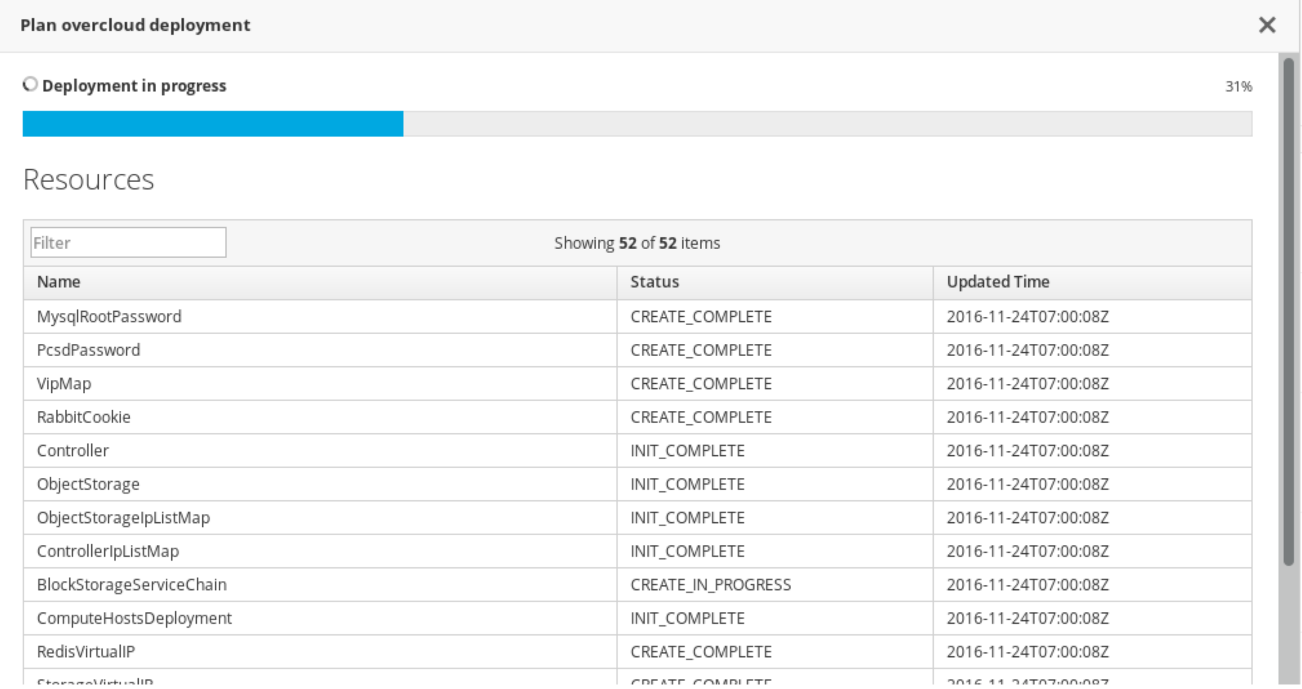
Not all pre-deployment validations have passed

It is highly recommended that you resolve all validation issues before continuing.

Are you sure you want to deploy this plan?

When you are ready to deploy, click **Deploy**.

The UI regularly monitors the progress of the overcloud's creation and display a progress bar indicating the current percentage of progress. The **View detailed information** link displays a log of the current OpenStack Orchestration stacks in your overcloud.



Deployment in progress 31%

Resources

Filter — Showing **52** of **52** items

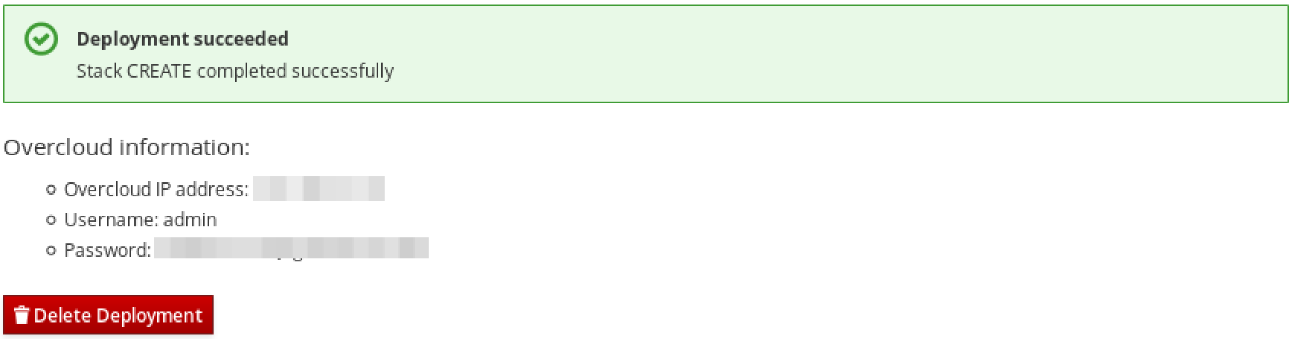
Name	Status	Updated Time
MysqlRootPassword	CREATE_COMPLETE	2016-11-24T07:00:08Z
PcsdPassword	CREATE_COMPLETE	2016-11-24T07:00:08Z
VipMap	CREATE_COMPLETE	2016-11-24T07:00:08Z
RabbitCookie	CREATE_COMPLETE	2016-11-24T07:00:08Z
Controller	INIT_COMPLETE	2016-11-24T07:00:08Z
ObjectStorage	INIT_COMPLETE	2016-11-24T07:00:08Z
ObjectStorageIpListMap	INIT_COMPLETE	2016-11-24T07:00:08Z
ControllerIpListMap	INIT_COMPLETE	2016-11-24T07:00:08Z
BlockStorageServiceChain	CREATE_IN_PROGRESS	2016-11-24T07:00:08Z
ComputeHostsDeployment	INIT_COMPLETE	2016-11-24T07:00:08Z
RedisVirtualIP	CREATE_COMPLETE	2016-11-24T07:00:08Z
StorageVirtualIP	CREATE_COMPLETE	2016-11-24T07:00:08Z

Wait until the overcloud deployment completes.

After the overcloud creation process completes, the **4 Deploy** section displays the current overcloud status and the following details:

- **IP address** - The IP address for accessing your overcloud.
- **Password** - The password for the OpenStack **admin** user on the overcloud.

Use this information to access your overcloud.



Deployment succeeded
Stack CREATE completed successfully

Overcloud information:

- Overcloud IP address: [redacted]
- Username: admin
- Password: [redacted]

6.10. COMPLETING THE OVERCLOUD CREATION

This concludes the creation of the overcloud through the director’s UI. For post-creation functions, see Chapter 7, *Performing Tasks after Overcloud Creation*.

CHAPTER 7. PERFORMING TASKS AFTER OVERCLOUD CREATION

This chapter explores some of the functions you perform after creating your overcloud of choice.

7.1. CREATING THE OVERCLOUD TENANT NETWORK

The overcloud requires a Tenant network for instances. Source the **overcloud** and create an initial Tenant network in Neutron. For example:

```
$ source ~/overcloudrc
$ openstack network create default
$ openstack subnet create default --network default --gateway 172.20.1.1 --subnet-range
172.20.0.0/16
```

This creates a basic Neutron network called **default**. The overcloud automatically assigns IP addresses from this network using an internal DHCP mechanism.

Confirm the created network with **neutron net-list**:

```
$ openstack network list
+-----------------------+-------------+--------------------------------------+
| id                    | name        | subnets                              |
+-----------------------+-------------+--------------------------------------+
| 95fadaa1-5dda-4777... | default     | 7e060813-35c5-462c-a56a-1c6f8f4f332f |
+-----------------------+-------------+--------------------------------------+
```

7.2. CREATING THE OVERCLOUD EXTERNAL NETWORK

You need to create the External network on the overcloud so that you can assign floating IP addresses to instances.

Using a Native VLAN

This procedure assumes a dedicated interface or native VLAN for the External network.

Source the **overcloud** and create an External network in Neutron. For example:

```
$ source ~/overcloudrc
$ openstack network create public --external --provider-network-type flat --provider-physical-network
datacentre
$ 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
```

In this example, you create a network with the name **public**. The overcloud requires this specific name for the default floating IP pool. This is also important for the validation tests in Section 7.6, "Validating the Overcloud".

This command also maps the network to the **datacentre** physical network. As a default, **datacentre** maps to the **br-ex** bridge. Leave this option as the default unless you have used custom neutron settings during the overcloud creation.

Using a Non-Native VLAN

If not using the native VLAN, assign the network to a VLAN using the following commands:

```
$ source ~/overcloudrc
$ openstack network create public --external --provider-network-type vlan --provider-physical-network
datacentre --provider-segment 104
$ 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
```

The **provider:segmentation_id** value defines the VLAN to use. In this case, you can use 104.

Confirm the created network with **neutron net-list**:

```
$ openstack network list
+-----------------------+-------------+--------------------------------------+
| id                    | name        | subnets                              |
+-----------------------+-------------+--------------------------------------+
| d474fe1f-222d-4e32... | public      | 01c5f621-1e0f-4b9d-9c30-7dc59592a52f |
+-----------------------+-------------+--------------------------------------+
```

7.3. CREATING ADDITIONAL FLOATING IP NETWORKS

Floating IP networks can use any bridge, not just **br-ex**, as long as you meet the following conditions:

- **NeutronExternalNetworkBridge** is set to **"''"** in your network environment file.
- You have mapped the additional bridge during deployment. For example, to map a new bridge called **br-floating** to the **floating** physical network:

```
$ openstack overcloud deploy --templates -e /usr/share/openstack-tripleo-heat-
templates/environments/network-isolation.yaml -e ~/templates/network-environment.yaml --
neutron-bridge-mappings datacentre:br-ex,floating:br-floating
```

Create the Floating IP network after creating the overcloud:

```
$ openstack network create ext-net --external --provider-physical-network floating --provider-network-
type vlan --provider-segment 105
$ openstack subnet create ext-subnet --network ext-net --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
```

7.4. CREATING THE OVERCLOUD PROVIDER NETWORK

A provider network is a network attached physically to a network existing outside of the deployed overcloud. This can be an existing infrastructure network or a network that provides external access directly to instances through routing instead of floating IPs.

When creating a provider network, you associate it with a physical network, which uses a bridge mapping. This is similar to floating IP network creation. You add the provider network to both the Controller and the Compute nodes because the Compute nodes attach VM virtual network interfaces directly to the attached network interface.

For example, if the desired provider network is a VLAN on the br-ex bridge, use the following command to add a provider network on VLAN 201:

```
$ openstack network create provider_network --provider-physical-network datacentre --provider-network-type vlan --provider-segment 201 --share
```

This command creates a shared network. It is also possible to specify a tenant instead of specifying **--share**. That network will only be available to the specified tenant. If you mark a provider network as external, only the operator may create ports on that network.

Add a subnet to a provider network if you want neutron to provide DHCP services to the tenant instances:

```
$ openstack subnet create provider-subnet --network  provider_network --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
```

Other networks might require access externally through the provider network. In this situation, create a new router so that other networks can route traffic through the provider network:

```
$ openstack router create external
$ openstack router set --external-gateway provider_network external
```

Attach other networks to this router. For example, if you had a subnet called **subnet1**, you can attach it to the router with the following commands:

```
$ openstack router add subnet external subnet1
```

This adds **subnet1** to the routing table and allows traffic using **subnet1** to route to the provider network.

7.5. CREATING A BASIC OVERCLOUD FLAVOR

Validation steps in this guide assume that your installation contains flavors. If you have not already created at least one flavor, use the following commands to create a basic set of default flavors that have a range of storage and processing capability:

```
$ 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
```

Command options

ram

Use the **ram** option to define the maximum RAM for the flavor.

disk

Use the **disk** option to define the hard disk space for the flavor.

vcpus

Use the **vcpus** option to define the quantity of virtual CPUs for the flavor.

Use **$ openstack flavor create --help** to learn more about the **openstack flavor create** command.

7.6. VALIDATING THE OVERCLOUD

The overcloud uses the OpenStack Integration Test Suite (tempest) tool set to conduct a series of integration tests. This section provides information on preparations for running the integration tests. For full instruction on using the OpenStack Integration Test Suite, see the OpenStack Integration Test Suite Guide.

Before Running the Integration Test Suite

If running this test from the undercloud, ensure that the undercloud host has access to the overcloud's Internal API network. 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
$ sudo ovs-vsctl add-port br-ctlplane vlan201 tag=201 -- set interface vlan201 type=internal
$ sudo ip l set dev vlan201 up; sudo ip addr add 172.16.0.201/24 dev vlan201
```

Before running OpenStack Validation, check that the **heat_stack_owner** role exists in your overcloud:

```
$ source ~/overcloudrc
$ openstack role list
+----------------------------------+------------------+
| ID                               | Name             |
+----------------------------------+------------------+
| 6226a517204846d1a26d15aae1af208f | swiftoperator    |
| 7c7eb03955e545dd86bbfeb73692738b | heat_stack_owner |
+----------------------------------+------------------+
```

If the role does not exist, create it:

```
$ openstack role create heat_stack_owner
```

After Running the Integration Test Suite

After completing the validation, remove any temporary connections to the overcloud's Internal API. In this example, use the following commands to remove the previously created VLAN on the undercloud:

```
$ source ~/stackrc
$ sudo ovs-vsctl del-port vlan201
```

7.7. FENCING THE CONTROLLER NODES

Fencing is the process of isolating a node to protect a cluster and its resources. Without fencing, a faulty node can cause data corruption in a cluster.

The director uses Pacemaker to provide a highly available cluster of Controller nodes. Pacemaker uses a process called STONITH (Shoot-The-Other-Node-In-The-Head) to help fence faulty nodes. By default, STONITH is disabled on your cluster and requires manual configuration so that Pacemaker can control the power management of each node in the cluster.

NOTE

Login to each node as the **heat-admin** user from the **stack** user on the director. The overcloud creation automatically copies the **stack** user's SSH key to each node's **heat-admin**.

Verify you have a running cluster with **pcs status**:

```
$ sudo pcs status
Cluster name: openstackHA
Last updated: Wed Jun 24 12:40:27 2015
Last change: Wed Jun 24 11:36:18 2015
Stack: corosync
Current DC: lb-c1a2 (2) - partition with quorum
Version: 1.1.12-a14efad
3 Nodes configured
141 Resources configured
```

Verify that stonith is disabled with **pcs property show**:

```
$ sudo pcs property show
Cluster Properties:
cluster-infrastructure: corosync
cluster-name: openstackHA
dc-version: 1.1.12-a14efad
have-watchdog: false
stonith-enabled: false
```

The Controller nodes contain a set of fencing agents for the various power management devices the director supports. This includes:

Table 7.1. Fence Agents

Device	Type
fence_ipmilan	The Intelligent Platform Management Interface (IPMI)
fence_idrac, **fence_drac5**	Dell Remote Access Controller (DRAC)
fence_ilo	Integrated Lights-Out (iLO)
fence_ucs	Cisco UCS - For more information, see Configuring Cisco Unified Computing System (UCS) Fencing on an OpenStack High Availability Environment
fence_xvm, **fence_virt**	Libvirt and SSH

The rest of this section uses the IPMI agent (**fence_ipmilan**) as an example.

View a full list of IPMI options that Pacemaker supports:

```
$ sudo pcs stonith describe fence_ipmilan
```

Each node requires configuration of IPMI devices to control the power management. This involves adding a **stonith** device to Pacemaker for each node. Use the following commands for the cluster:

NOTE

The second command in each example is to prevent the node from asking to fence itself.

For Controller node 0:

```
$ sudo pcs stonith create my-ipmilan-for-controller-0 fence_ipmilan pcmk_host_list=overcloud-controller-0 ipaddr=192.0.2.205 login=admin passwd=p@55w0rd! lanplus=1 cipher=1 op monitor interval=60s
$ sudo pcs constraint location my-ipmilan-for-controller-0 avoids overcloud-controller-0
```

For Controller node 1:

```
$ sudo pcs stonith create my-ipmilan-for-controller-1 fence_ipmilan pcmk_host_list=overcloud-controller-1 ipaddr=192.0.2.206 login=admin passwd=p@55w0rd! lanplus=1 cipher=1 op monitor interval=60s
$ sudo pcs constraint location my-ipmilan-for-controller-1 avoids overcloud-controller-1
```

For Controller node 2:

```
$ sudo pcs stonith create my-ipmilan-for-controller-2 fence_ipmilan pcmk_host_list=overcloud-controller-2 ipaddr=192.0.2.207 login=admin passwd=p@55w0rd! lanplus=1 cipher=1 op monitor interval=60s
$ sudo pcs constraint location my-ipmilan-for-controller-2 avoids overcloud-controller-2
```

Run the following command to see all stonith resources:

```
$ sudo pcs stonith show
```

Run the following command to see a specific stonith resource:

```
$ sudo pcs stonith show [stonith-name]
```

Finally, enable fencing by setting the **stonith** property to **true**:

```
$ sudo pcs property set stonith-enabled=true
```

Verify the property:

```
$ sudo pcs property show
```

7.8. MODIFYING THE OVERCLOUD ENVIRONMENT

Sometimes you might intend to modify the overcloud to add additional features, or change the way it operates. 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 5.6, “Creating the Overcloud with the CLI Tools”, you would rerun the following command:

```
$ openstack overcloud deploy --templates -e /usr/share/openstack-tripleo-heat-
templates/environments/network-isolation.yaml -e ~/templates/network-environment.yaml -e
~/templates/storage-environment.yaml --control-scale 3 --compute-scale 3 --ceph-storage-scale 3 --
control-flavor control --compute-flavor compute --ceph-storage-flavor ceph-storage --ntp-server
pool.ntp.org --neutron-network-type vxlan --neutron-tunnel-types vxlan
```

The director checks the **overcloud** stack in heat, and then updates each item in the stack with the environment files and heat templates. It does not recreate the overcloud, but rather changes the existing overcloud.

If you aim to include a new environment file, add it to the **openstack overcloud deploy** command with a **-e** option. For example:

```
$ openstack overcloud deploy --templates -e /usr/share/openstack-tripleo-heat-
templates/environments/network-isolation.yaml -e ~/templates/network-environment.yaml -e
~/templates/storage-environment.yaml -e ~/templates/new-environment.yaml --control-scale 3 --
compute-scale 3 --ceph-storage-scale 3 --control-flavor control --compute-flavor compute --ceph-
storage-flavor ceph-storage --ntp-server pool.ntp.org --neutron-network-type vxlan --neutron-tunnel-
types vxlan
```

This includes the new parameters and resources from the environment file into the stack.

IMPORTANT

It is advisable not to make manual modifications to the overcloud’s configuration as the director might overwrite these modifications later.

7.9. IMPORTING VIRTUAL MACHINES INTO THE OVERCLOUD

Use the following procedure if you have an existing OpenStack environment and aim to migrate its virtual machines to your Red Hat OpenStack Platform environment.

Create a new image by taking a snapshot of a running server and download the image.

```
$ openstack server image create instance_name --name image_name
$ openstack image save image_name --file exported_vm.qcow2
```

Upload the exported image into the overcloud and launch a new instance.

```
$ openstack image create imported_image --file exported_vm.qcow2 --disk-format qcow2 --
container-format bare
$ openstack server create  imported_instance --key-name default --flavor m1.demo --image
imported_image --nic net-id=net_id
```

IMPORTANT

Each VM disk has to be copied from the existing OpenStack environment and into the new Red Hat OpenStack Platform. Snapshots using QCOW will lose their original layering system.

7.10. RUNNING ANSIBLE AUTOMATION

The director provides the ability to run Ansible-based automation on your OpenStack Platform environment. The director uses the **tripleo-ansible-inventory** command to generate a dynamic inventory of nodes in your environment.

> IMPORTANT
>
> The dynamic inventory tool only includes the undercloud and the default **controller** and **compute** overcloud nodes. Other roles are not supported.

To view a dynamic inventory of nodes, run the **tripleo-ansible-inventory** command after sourcing **stackrc**:

```
$ souce ~/stackrc
$ tripleo-ansible-inventory --list
```

The **--list** option provides details on all hosts.

This outputs the dynamic inventory in a JSON format:

```
{"overcloud": {"children": ["controller", "compute"], "vars": {"ansible_ssh_user": "heat-admin"}},
"controller": ["192.0.2.2"], "undercloud": {"hosts": ["localhost"], "vars": {"overcloud_horizon_url":
"http://192.0.2.4:80/dashboard", "overcloud_admin_password": "abcdefghijklm12345678",
"ansible_connection": "local"}}, "compute": ["192.0.2.3"]}
```

To execute Ansible automations on your environment, run the **ansible** command and include the full path of the dynamic inventory tool using the **-i** option. For example:

```
ansible [HOSTS] -i /bin/tripleo-ansible-inventory [OTHER OPTIONS]
```

- Exchange **[HOSTS]** for the type of hosts to use. For example:
  - **controller** for all Controller nodes
  - **compute** for all Compute nodes
  - **overcloud** for all overcloud child nodes i.e. **controller** and **compute**
  - **undercloud** for the undercloud
  - **"*"** for all nodes
- Exchange **[OTHER OPTIONS]** for the additional Ansible options. Some useful options include:
  - **--ssh-extra-args='-o StrictHostKeyChecking=no'** to bypasses confirmation on host key checking.
  - **-u [USER]** 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.
  - **-m [MODULE]** to use a specific Ansible module. The default is **command**, which executes Linux commands.

- **-a [MODULE_ARGS]** to define arguments for the chosen module.

IMPORTANT

Ansible automation on the overcloud falls outside the standard overcloud stack. This means subsequent execution of the **openstack overcloud deploy** command might override Ansible-based configuration for OpenStack Platform services on overcloud nodes.

7.11. PROTECTING THE OVERCLOUD FROM REMOVAL

To avoid accidental removal of the overcloud with the **heat stack-delete overcloud** command, Heat contains a set of policies to restrict certain actions. Edit the **/etc/heat/policy.json** and find the following parameter:

```
"stacks:delete": "rule:deny_stack_user"
```

Change it to:

```
"stacks:delete": "rule:deny_everybody"
```

Save the file.

This prevents removal of the overcloud with the **heat** client. To allow removal of the overcloud, revert the policy to the original value.

7.12. REMOVING THE OVERCLOUD

The whole overcloud can be removed when desired.

Delete any existing overcloud:

```
$ openstack stack delete overcloud
```

Remove the overcloud plan from the director:

```
$ openstack overcloud plan delete overcloud
```

Confirm the deletion of the overcloud:

```
$ openstack stack list
```

Deletion takes a few minutes.

Once the removal completes, follow the standard steps in the deployment scenarios to recreate your overcloud.

CHAPTER 8. MIGRATING VIRTUAL MACHINES BETWEEN COMPUTE NODES

In some situations, you might need to migrate virtual machines from one Compute node to another Compute node in the overcloud. For example:

- **Compute Node Maintenance:** If you must temporarily take a Compute node out of service, you can temporarily migrate virtual machines running on the Compute node to another Compute node. Common scenarios include hardware maintenance, hardware repair, kernel upgrades and software updates.

- **Failing Compute Node:** If a Compute node is about to fail and must be serviced or replaced, you must migrate virtual machines from the failing Compute node to a healthy Compute node. For Compute nodes that have already failed, see Evacuating VMs.

- **Workload Rebalancing:** You can consider migrating one or more virtual machines to another Compute node to rebalance the workload. For example, you can consolidate virtual machines on a Compute node to conserve power, migrate virtual machines to a Compute node that is physically closer to other networked resources to reduce latency, or distribute virtual machines across Compute nodes to avoid hot spots and increase resiliency.

The director configures **all** Compute nodes to provide secure migration. All Compute nodes also require a shared SSH key to provide each host's **nova** user with access to other Compute nodes during the migration process. The director creates this key using the **OS::TripleO::Services::NovaCompute** composable service. This composable service is one of the main services included on all Compute roles by default (see Composable Services and Custom Roles in **Advanced Overcloud Customization**).

UPDATES TO MIGRATION FUNCTIONALITY

The latest update of Red Hat OpenStack Platform 10 includes the **OS::TripleO::Services::Sshd** composable service, which is required for live migration capabilities. The director's core template collection did not include this service in the initial release but is now included in the **openstack-tripleo-heat-templates-5.2.0-12** package and later versions.

- If using the default roles, update your environment to use the Heat templates from the **openstack-tripleo-heat-templates-5.2.0-12** package or later versions.

- If using a custom roles data file, update your environment to use the Heat templates from the **openstack-tripleo-heat-templates-5.2.0-12** package or later versions, include the **OS::TripleO::Services::Sshd** service on each overcloud role, then update your overcloud stack to include the new service.

For more information, see "Red Hat OpenStack Platform director (TripleO) CVE-2017-2637 bug and Red Hat OpenStack Platform".

IMPORTANT

RHSA-2018:0369 - Security Advisory issued on 2018-02-27 and RHBA-2019:0074 - Bug Fix Advisory issued on 2019-01-16 impact the migration of virtual machines.

RHSA-2018:0369 - Security Advisory prevents live migration of virtual machines to Compute nodes where the CPUs defined in the domain XML are already in use. RHSA-2018:0369 - Security Advisory unnecessarily imposes the same restrictions on operations such as cold migration, evacuation, resize and unshelve. RHBA-2019:0074 - Bug Fix Advisory relaxes those restrictions by making them optional. See the new **cpu_pinning_migration_quick_fail** configuration option introduced with this advisory for details.

Red Hat recommends applying both advisories.

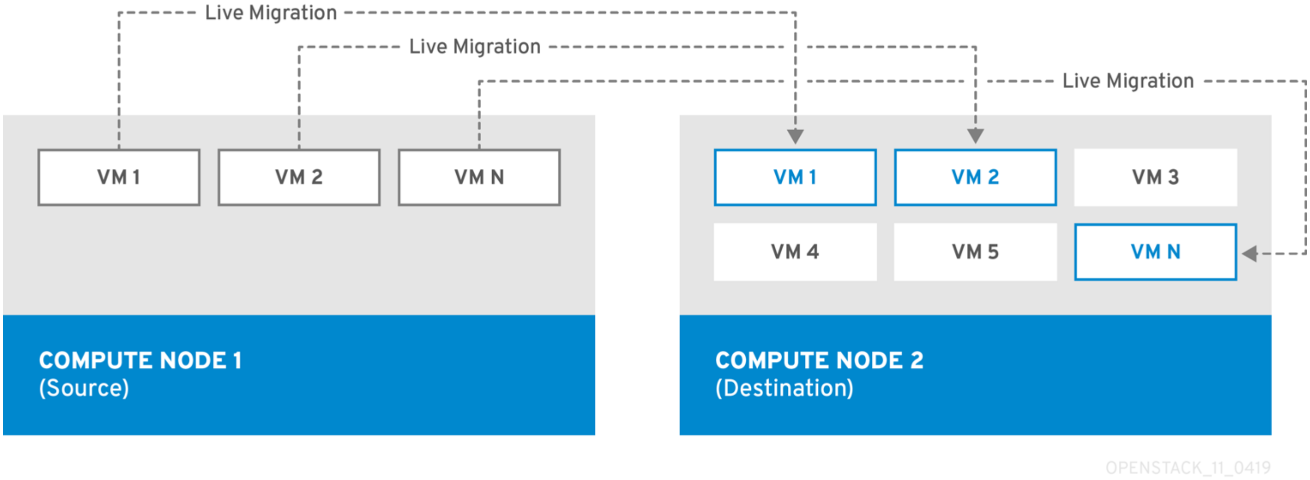
See Migration Constraints for additional details on migration constraints.

8.1. MIGRATION TYPES

OpenStack Platform supports two types of migration:

Live Migration

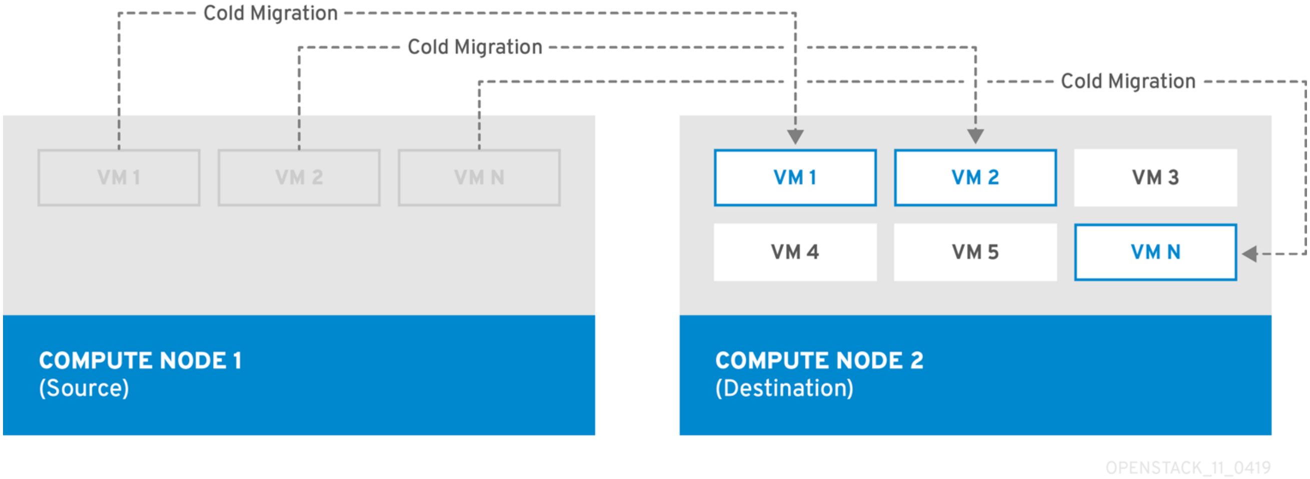
Live migration involves spinning up the virtual machine on the destination node and shutting down the virtual machine on the source node seamlessly while maintaining state consistency.



Live migration handles virtual machine migration with little or no perceptible downtime. In some cases, virtual machines **cannot** use live migration. See Migration Constraints for details on migration constraints.

Cold Migration

Cold migration or non-live migration involves **nova** shutting down a virtual machine before migrating it from the source Compute node to the destination Compute node.



Cold migration involves some downtime for the virtual machine. However, cold migration still provides the migrated virtual machine with access to the same volumes and IP addresses.

IMPORTANT

For source Compute nodes that have already failed, see Evacuation. Migration requires that both the source and destination Compute nodes are running.

8.2. MIGRATION CONSTRAINTS

In some cases, migrating virtual machines involves additional constraints. Migration constraints typically arise with block migration, configuration disks, or when one or more virtual machines access physical hardware on the Compute node.

CPU Constraints

The source and destination Compute nodes **must** have the same CPU architecture. For example, Red Hat does **not support** migrating a virtual machine from an **x86_64** CPU to a **ppc64le** CPU. In some cases, the CPU of the source and destination Compute node **must** match exactly, such as virtual machines using CPU host passthrough. In all cases, the CPU features of the destination node **must** be a superset of the CPU features on the source node. Additionally, when virtual machines use CPU pinning, the NUMA node used on the source node must target the same NUMA node on the destination Compute node and the NUMA node must be empty.

Memory Constraints

The destination Compute node **must** have sufficient available RAM. Memory oversubscription **can** cause migration to fail. Additionally, virtual machines using a NUMA topology must have sufficient available RAM on the same NUMA node on the destination Compute node.

Block Migration Constraints

Migrating virtual machines that use disks stored locally on a Compute node takes significantly longer than migrating volume-backed virtual machines that utilize shared storage such as Red Hat Ceph Storage. This latency arises because **nova** migrates local disks block-by-block between the Compute nodes over the control plane network by default. By contrast, volume-backed instances using shared storage such as Red Hat Ceph Storage do not have to migrate the volumes, because each Compute node already has access to the shared storage.

NOTE

Network congestion in the control plane network caused by migrating local disks or virtual machines that consume large amounts of RAM could impact the performance of other systems using the control plane network, such as RabbitMQ.

Read-only Drive Migration Constraints

Migrating a drive is **only** supported if the drive has both read *and* write capabilities. For example, **nova** cannot migrate a CD-ROM drive or a read-only config drive. However, **nova** can migrate a drive with *both* read and write capabilities, including a config drive with a drive format such as **vfat**.

Live Migration Constraints

There are a few additional live migration constraints in Red Hat OpenStack Platform:

- **No New Operations During Migration:** To achieve state consistency between the copies of the virtual machine on the source and destination nodes, Red Hat OpenStack Platform must prevent new operations during live migration. Otherwise, live migration could take a long time or potentially never end if writes to memory occur faster than live migration can replicate the state of the memory.

- **Non-Uniform Memory Access (NUMA):** In previous releases, migrating virtual machines when configured with a NUMA topology was not recommended. Currently, **nova** can migrate virtual machines with NUMA topology cleanly subject to some constraints.

- **CPU Pinning:** When a flavor uses CPU pinning, the flavor implicitly introduces a NUMA topology to the virtual machine and maps its CPUs and memory to specific host CPUs and memory. The difference been a simple NUMA topology and CPU pinning is that NUMA uses a *range* of CPU cores, whereas CPU pinning uses specific CPU cores. See CPU pinning for additional details.

- **Data Plane Development Kit (DPDK):** When a virtual machine uses DPDK, such as a virtual machine running Open vSwitch with **dpdk-netdev**, the virtual machine also uses huge pages which imposes a NUMA topology such that **nova** pins the virtual machine to a NUMA node.

nova can live migrate a virtual machine that uses NUMA, CPU pinning or DPDK. However, the destination Compute node **must** have sufficient capacity **on the same NUMA node** that the virtual machine uses on the source Compute node. For example, if a virtual machine uses **NUMA 0** on **overcloud-compute-0**, when migrating the virtual machine to **overcloud-compute-1**, you must ensure that **overcloud-compute-1** has sufficient capacity on **NUMA 0** to support the virtual machine in order to use live migration.

Constraints that Preclude Live Migration

There are a few cases where virtual machine configuration precludes live migration in Red Hat OpenStack Platform:

- **Single-root Input/Output Virtualization (SR-IOV):** You can assign SR-IOV Virtual Functions (VFs) to virtual machines. However, this prevents live migration. Unlike a regular network device, an SR-IOV VF network device does not have a permanent unique MAC address. The VF network device receives a new MAC address each time the Compute node reboots or when **nova-scheduler** migrates the virtual machine to a new Compute node. Consequently, **nova** cannot live migrate virtual machines that use SR-IOV in OpenStack Platform 10. You must cold migrate virtual machines that use SR-IOV.

- **PCI Passthrough:** QEMU/KVM hypervisors support attaching PCI devices on the Compute node to a virtual machine. PCI passthrough allows a virtual machine to have exclusive access to

PCI devices, which appear and behave as if they are physically attached to the virtual machine's operating system. However, since PCI passthrough involves physical addresses, **nova** does not support live migration of virtual machines using PCI passthrough in OpenStack Platform 10.

8.3. PRE-MIGRATION PROCEDURES

Before migrating one or more virtual machines, perform the following steps:

Procedure

1. From the undercloud, identify the source Compute node hostname and the destination Compute node hostname.

```
$ source ~/overcloudrc
$ openstack compute service list
```

2. List virtual machines on the source Compute node and locate the ID of the virtual machine(s) you intend to migrate:

```
$ openstack server list --host [source] --all-projects
```

Replace **[source]** with the host name of the source Compute node.

Pre-Migration Procedure for Compute Node Maintenance

If you are taking down the source Compute node for maintenance, disable the source Compute node from the undercloud to ensure that the scheduler does not attempt to assign new virtual machines to the source Compute node during maintenance.

```
$ openstack compute service set [source] nova-compute --disable
```

Replace **[source]** with the host name of the source Compute node.

Pre-Migration Procedure for NUMA, CPU-pinned and DPDK Instances

When migrating virtual machines that use NUMA, CPU-pinning or DPDK, the destination Compute node should have an identical hardware specification and configuration as the source Compute node. Additionally, the destination Compute node should have no virtual machines running on it to ensure that it preserves the NUMA topology of the source Compute node.

> NOTE
>
> When migrating virtual machines using NUMA, CPU-pinning or DPDK, the **/etc/nova/nova.conf** file requires appropriate values for the **scheduler_default_filters** configuration setting, such as **AggregateInstanceExtraSpecFilter** and **NUMATopologyFilter**. You can accomplish this by setting the **NovaSchedulerDefaultFilters** heat parameter in an environment file.

1. If the destination Compute node for NUMA, CPU-pinned or DPDK virtual machines is **not** disabled, disable it to prevent the scheduler from assigning virtual machines to the node.

```
$ openstack compute service set [dest] nova-compute --disable
```

Replace **[dest]** with the host name of the destination Compute node.

2. Ensure the destination Compute node has no virtual machines, except for virtual machines previously migrated from the source Compute node when migrating multiple DPDK or NUMA virtual machines.

   ```
   $ openstack server list --host [dest] --all-projects
   ```

   Replace **[dest]** with the host name of the destination Compute node.

3. Ensure the destination Compute node has sufficient resources to run the NUMA, CPU-pinned or DPDK virtual machine.

   ```
   $ openstack host show overcloud-compute-n
   $ ssh overcloud-compute-n
   $ numactl --hardware
   $ exit
   ```

   Replace **overcloud-compute-n** with the host name of the destination Compute node.

4. To discover NUMA information about the source or destination Compute nodes, execute the following as needed:

   ```
   $ ssh root@overcloud-compute-n
   # lscpu && lscpu | grep NUMA
   # virsh nodeinfo
   # virsh capabilities
   # exit
   ```

   Use **ssh** to connect to **overcloud-compute-n** where **overcloud-compute-n** is the source or destination Compute node.

5. If you are unsure if a virtual machine uses NUMA, check the flavor of the virtual machine.

   ```
   $ openstack server list -c Name -c Flavor --name [vm]
   ```

   Replace **[vm]** with the name or ID for the virtual machine.

   Then, check the flavor:

   ```
   $ openstack flavor show [flavor]
   ```

   Replace **[flavor]** with the name or ID of the flavor. If the result of the **properties** field includes **hw:mem_page_size** with a value other than **any** such as **2MB**, **2048** or **1GB**, the virtual machine will have a NUMA topology. If the **properties** field includes **aggregate_instance_extra_specs:pinned='true'**, the virtual machine uses CPU pinning. If the **properties** field includes **hw:numa_nodes**, **nova** restricts the virtual machine to a specific NUMA node.

6. For each virtual machine using NUMA, consider retrieving information about the NUMA topology from the underlying Compute node so that you can verify that the NUMA topology on the destination Compute node reflects the NUMA topology of the source Compute node after migration is complete.

   ```
   $ ssh root@overcloud-compute-n
   # virsh vcpuinfo [vm]
   # virsh numatune [vm]
   ```

```
# exit
```

Replace **[vm]** with the name of the virtual machine. The **vcpuinfo** command provides details about NUMA and CPU pinning. The **numatune** command provides details about which NUMA node the virtual machine is using.

8.4. LIVE MIGRATE A VIRTUAL MACHINE

Live migration moves a virtual machine from a source Compute node to a destination Compute node with a minimal amount of downtime. However, live migration might not be appropriate for all virtual machines. See Migration Constraints for additional details.

Procedure

1. To live migrate a virtual machine, specify the virtual machine and the destination Compute node:

   ```
   $ openstack server migrate [vm] --live [dest] --wait
   ```

   Replace **[vm]** with the name or ID of the virtual machine. Replace **[dest]** with the hostname of the destination Compute node. Specify the **--block-migration** flag if migrating a locally stored volume.

2. Wait for migration to complete. See Check Migration Status to check the status of the migration.

3. Confirm the migration was successful:

   ```
   $ openstack server list --host [dest] --all-projects
   ```

   Replace **[dest]** with the hostname of the destination Compute node.

4. For virtual machines using NUMA, CPU-pinning or DPDK, consider retrieving information about the NUMA topology from a Compute node to compare it with NUMA topology retrieved during the pre-migration procedure.

   ```
   $ ssh root@overcloud-compute-n
   # virsh vcpuinfo [vm]
   # virsh numatune [vm]
   # exit
   ```

   Replace **overcloud-compute-n** with the host name of the Compute node. Replace **[vm]** with the name of the virtual machine. Comparing the NUMA topologies of the source and destination Compute nodes helps to ensure that the source and destination Compute nodes use the same NUMA topology.

5. Repeat this procedure for each additional virtual machine that you intend to migrate.

When you have finished migrating the virtual machines, proceed to the Post-migration Procedures.

8.5. COLD MIGRATE A VIRTUAL MACHINE

Cold migrating a virtual machine involves stopping the virtual machine and moving it to another Compute node. Cold migration facilitates migration scenarios that live migrating cannot facilitate, such as migrating virtual machines using PCI passthrough or Single-Root Input/Output Virtualization (SR-IOV). The Scheduler automatically selects the destination Compute node. See Migration Constraints for additional details.

Procedure

1. To migrate a virtual machine, specify the virtual machine.

   ```
   $ openstack server migrate [vm] --wait
   ```

   Replace **[vm]** with the virtual machine ID. Specify the **--block-migration** flag if migrating a locally stored volume.

2. Wait for migration to complete. See Check Migration Status to check the status of the migration.

3. Confirm the migration was successful.

   ```
   $ openstack server list --all-projects
   ```

When you have finished migrating virtual machines, proceed to the Post-migration Procedures.

8.6. CHECK MIGRATION STATUS

Migration involves numerous state transitions before migration is complete. During a healthy migration, the migration state typically transitions as follows:

1. **Queued: nova** accepted the request to migrate a virtual machine and migration is pending.

2. **Preparing: nova** is preparing to migrate the virtual machine.

3. **Running: nova** is in the process of migrating the virtual machine.

4. **Post-migrating: nova** has built the virtual machine on the destination Compute node and is freeing up resources on the source Compute node.

5. **Completed: nova** has completed migrating the virtual machine and finished freeing up resources on the source Compute node.

Procedure

1. Retrieve the list of migrations for the virtual machine.

   ```
   $ nova server-migration-list [vm]
   ```

   Replace **[vm]** with the virtual machine name or ID.

2. Show the status of the migration.

   ```
   $ nova server-migration-show [vm] [migration]
   ```

   Replace **[vm]** with the virtual machine name or ID. Replace **[migration]** with the ID of the migration.

Sometimes virtual machine migration can take a long time or encounter errors. See Section 8.8, "Troubleshooting Migration" for details.

8.7. POST-MIGRATION PROCEDURES

After migrating one or more virtual machines, review the following procedures and execute them as appropriate.

Post-Migration Procedure for Compute Node Maintenance

If you previously shut down the source Compute node for maintenance and maintenance is complete, you may re-enable the source Compute node from the undercloud to ensure that the scheduler can assign new virtual machines to the source Compute node.

```
$ source ~/overcloudrc
$ openstack compute service set [source] nova-compute --enable
```

Replace **[source]** with the host name of the source Compute node.

Post-Migration Procedure for NUMA, CPU-pinned or DPDK Instances

After migrating virtual machines that use NUMA, CPU-pinning or DPDK, you may re-enable the destination Compute node from the undercloud.

```
$ source ~/overcloudrc
$ openstack compute service set [dest] nova-compute --enable
```

Replace **[dest]** with the host name of the destination Compute node.

8.8. TROUBLESHOOTING MIGRATION

There are several issues that can arise during virtual machine migration:

1. The migration process encounters errors.
2. The migration process never ends.
3. Virtual machine performance degrades after migration

Errors During Migration

The following issues can send the migration operation into an **error** state:

1. Running a cluster with different versions of OpenStack.
2. Specifying a virtual machine ID that cannot be found.
3. The virtual machine you are trying to migrate is in an **error** state.
4. The Compute service is shutting down.
5. A race condition occurs.
6. Live migration enters a **failed** state.

When live migration enters a **failed** state, it is typically followed by an **error** state. The following common issues can cause a **failed** state:

1. A destination Compute host is not available.

2. A scheduler exception occurs.

3. The rebuild process fails due to insufficient computing resources.

4. A server group check fails.

5. The virtual machine on the source Compute node gets deleted before migration to the destination Compute node is complete.

Never-ending Live Migration

Live migration can fail to complete in a timely manner, which leaves migration in a perpetual **running** state. A common reason for a live migration that never completes is that client requests to the virtual machine running on the source Compute node create changes that occur faster than nova can replicate them to the destination Compute node.

There are a few ways to address this situation:

1. Abort the live migration.

2. Force the live migration to complete.

Aborting Live Migration

If the virtual machine state changes faster than the migration procedure can copy it to the destination node and you do not want to temporarily suspend the virtual machine's operations, you can abort the live migration procedure.

1. Retrieve the list of migrations for the virtual machine:

   ```
   $ nova server-migration-list [vm]
   ```

   Replace **[vm]** with the virtual machine name or ID.

2. Abort the live migration:

   ```
   $ nova live-migration-abort [vm] [migration]
   ```

   Replace **[vm]** with the virtual machine name or ID, and **[migration]** with the ID of the migration.

Forcing Live Migration to Complete

If the virtual machine state changes faster than the migration procedure can copy it to the destination node and you want to temporarily suspend the virtual machine's operations to force migration to complete, you can force the live migration procedure to complete.

IMPORTANT

Forcing live migration to complete might lead to perceptible downtime.

1. Retrieve the list of migrations for the virtual machine:

```
$ nova server-migration-list [vm]
```

Replace **[vm]** with the virtual machine name or ID.

2. Force the live migration to complete:

   ```
   $ nova live-migration-force-complete [vm] [migration]
   ```

   Replace **[vm]** with the virtual machine name or ID. Replace **[migration]** with the ID of the migration.

Virtual Machine Performance Degrades After Migration

For VMs using a NUMA topology, the source and destination Compute nodes must have the same NUMA topology and configuration. The destination Compute node's NUMA topology must have sufficient resources available. If the NUMA configuration between the source and destination Compute nodes is not the same, it is possible that live migration succeeds while the virtual machine performance degrades. For example, if the source Compute node maps NIC 1 to NUMA node 0, but the destination Compute node maps NIC 1 to NUMA node 5, after migration the virtual machine might route network traffic from a first CPU across the bus to a second CPU with NUMA node 5 to route traffic to NIC 1—resulting in expected behavior, but degraded performance. Similarly, if NUMA node 0 on the source Compute node has sufficient available CPU and RAM, but NUMA node 0 on the destination Compute node already has virtual machines using some of the resources, the virtual machine might run properly but suffer performance degradation. See Section 8.2, "Migration Constraints" for additional details.

CHAPTER 9. SCALING THE OVERCLOUD

WARNING

Do not use **openstack server delete** to remove nodes from the overcloud. Read the procedures defined in this section to properly remove and replace nodes.

There might be situations where you need to add or remove nodes after the creation of the overcloud. For example, you might need to add more Compute nodes to the overcloud. This situation requires updating the overcloud.

Use the following table to determine support for scaling each node type:

Table 9.1. Scale Support for Each Node Type

Node Type	Scale Up?	Scale Down?	Notes
Controller	N	N	
Compute	Y	Y	
Ceph Storage Nodes	Y	N	You must have at least 1 Ceph Storage node from the initial overcloud creation.
Block Storage Nodes	N	N	
Object Storage Nodes	Y	Y	Requires manual ring management, which is described in Section 9.6, “Replacing Object Storage Nodes”.

IMPORTANT

Make sure to leave at least 10 GB free space before scaling the overcloud. This free space accommodates image conversion and caching during the node provisioning process.

9.1. ADDING ADDITIONAL NODES

To add more nodes to the director’s node pool, create a new JSON file (for example, **newnodes.json**) containing the new node details to register:

```
{
  "nodes":[
    {
      "mac":[
        "dd:dd:dd:dd:dd:dd"
      ],
      "cpu":"4",
      "memory":"6144",
      "disk":"40",
      "arch":"x86_64",
      "pm_type":"pxe_ipmitool",
      "pm_user":"admin",
      "pm_password":"p@55w0rd!",
      "pm_addr":"192.0.2.207"
    },
    {
      "mac":[
        "ee:ee:ee:ee:ee:ee"
      ],
      "cpu":"4",
      "memory":"6144",
      "disk":"40",
      "arch":"x86_64",
      "pm_type":"pxe_ipmitool",
      "pm_user":"admin",
      "pm_password":"p@55w0rd!",
      "pm_addr":"192.0.2.208"
    }
  ]
}
```

See Section 5.1, “Registering Nodes for the Overcloud” for an explanation of these parameters.

Run the following command to register these nodes:

```
$ openstack baremetal import --json newnodes.json
```

After registering the new nodes, launch the introspection process for them. Use the following commands for each new node:

```
$ openstack baremetal node manage [NODE UUID]
$ openstack overcloud node introspect [NODE UUID] --provide
```

This detects and benchmarks the hardware properties of the nodes.

After the introspection process completes, tag each new node for its desired role. For example, for a Compute node, use the following command:

```
$ openstack baremetal node set --property capabilities='profile:compute,boot_option:local' [NODE
UUID]
```

Scaling the overcloud requires running the **openstack overcloud deploy** again with the desired number of nodes for a role. For example, to scale to 5 Compute nodes:

```
$ openstack overcloud deploy --templates --compute-scale 5 [OTHER_OPTIONS]
```

This updates the entire overcloud stack. Note that this only updates the stack. It does not delete the overcloud and replace the stack.

IMPORTANT

Make sure to include all environment files and options from your initial overcloud creation. This includes the same scale parameters for non-Compute nodes.

9.2. REMOVING COMPUTE NODES

There might be situations where you need to remove Compute nodes from the overcloud. For example, you might need to replace a problematic Compute node.

IMPORTANT

Before removing a Compute node from the overcloud, migrate the workload from the node to other Compute nodes. See Chapter 8, *Migrating Virtual Machines Between Compute Nodes* for more details.

Next, disable the node's Compute service on the overcloud. This stops the node from scheduling new instances.

```
$ source ~/overcloudrc
$ openstack compute service list
$ openstack compute service set [hostname] nova-compute --disable
$ source ~/stackrc
```

Removing overcloud nodes requires an update to the **overcloud** stack in the director using the local template files. First identify the UUID of the overcloud stack:

```
$ openstack stack list
```

Identify the UUIDs of the nodes to delete:

```
$ openstack server list
```

Run the following commands to update the overcloud plan and delete the nodes from the stack:

```
$ openstack overcloud deploy --update-plan-only \
  --templates  \
  -e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml \
  -e /home/stack/templates/network-environment.yaml \
  -e /home/stack/templates/storage-environment.yaml \
  -e /home/stack/templates/rhel-registration/environment-rhel-registration.yaml \
  [-e |...]
$ openstack overcloud node delete --stack [STACK_UUID] --templates -e [ENVIRONMENT_FILE]
[NODE1_UUID] [NODE2_UUID] [NODE3_UUID]
```

IMPORTANT

If you passed any extra environment files when you created the overcloud, pass them here again using the **-e** or **--environment-file** option to avoid making undesired manual changes to the overcloud.

IMPORTANT

Make sure the **openstack overcloud node delete** command runs to completion before you continue. Use the **openstack stack list** command and check the **overcloud** stack has reached an **UPDATE_COMPLETE** status.

Finally, remove the node's Compute service:

```
$ source ~/overcloudrc
$ openstack compute service list
$ openstack compute service delete [service-id]
$ source ~/stackrc
```

And remove the node's Open vSwitch agent:

```
$ source ~/overcloudrc
$ openstack network agent list
$ openstack network agent delete [openvswitch-agent-id]
$ source ~/stackrc
```

You are now free to remove the node from the overcloud and re-provision it for other purposes.

9.3. REPLACING COMPUTE NODES

If a Compute node fails, you can replace the node with a working one. Replacing a Compute node uses the following process:

- Migrate workload off the existing Compute node and shutdown the node. See Chapter 8, *Migrating Virtual Machines Between Compute Nodes* for this process.
- Remove the Compute node from the overcloud. See Section 9.2, "Removing Compute Nodes" for this process.
- Scale out the overcloud with a new Compute node. See Section 9.1, "Adding Additional Nodes" for this process.

This process ensures that a node can be replaced without affecting the availability of any instances.

9.4. 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. This also includes ensuring the node connects to the other nodes in the cluster.

This section provides instructions on how to replace a Controller node. The process involves running the **openstack overcloud deploy** command to update the overcloud with a request to replace a controller node. Note that this process is not completely automatic; during the overcloud stack update process, the **openstack overcloud deploy** command will at some point report a failure and halt the overcloud stack update. At this point, the process requires some manual intervention. Then the **openstack overcloud deploy** process can continue.

IMPORTANT

The following procedure only applies to high availability environments. Do not use this procedure if only using one Controller node.

9.4.1. Preliminary Checks

Before attempting to 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.

1. Check the current status of the **overcloud** stack on the undercloud:

   ```
   $ source stackrc
   $ openstack stack list --nested
   ```

   The **overcloud** stack and its subsequent child stacks should have either a **CREATE_COMPLETE** or **UPDATE_COMPLETE**.

2. 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
   $ sudo systemctl stop openstack-ironic-api.service openstack-ironic-conductor.service
   openstack-ironic-inspector.service openstack-ironic-inspector-dnsmasq.service
   $ sudo cp /var/lib/ironic-inspector/inspector.sqlite /home/stack/backup
   $ sudo systemctl start openstack-ironic-api.service openstack-ironic-conductor.service
   openstack-ironic-inspector.service openstack-ironic-inspector-dnsmasq.service
   ```

3. Check your undercloud contains 10 GB free storage to accommodate for image caching and conversion when provisioning the new node.

4. 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 get the Pacemaker status:

   ```
   $ ssh heat-admin@192.168.0.47 'sudo pcs status'
   ```

   The output should show all services running on the existing nodes and stopped on the failed node.

5. Check the following parameters on each node of the overcloud's MariaDB cluster:

   - **wsrep_local_state_comment: Synced**
   - **wsrep_cluster_size: 2**
     Use the following command to check these parameters on each running Controller node (respectively using 192.168.0.47 and 192.168.0.46 for IP addresses):

```
$ for i in 192.168.0.47 192.168.0.46 ; do echo "*** $i ***" ; ssh heat-admin@$i "sudo
mysql --exec=\"SHOW STATUS LIKE 'wsrep_local_state_comment'\" ; sudo mysql --
exec=\"SHOW STATUS LIKE 'wsrep_cluster_size'\""; done
```

6. 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 get the status

   ```
   $ ssh heat-admin@192.168.0.47 "sudo rabbitmqctl cluster_status"
   ```

   The **running_nodes** key should only show the two available nodes and not the failed node.

7. Disable fencing, if enabled. For example, if 192.168.0.47 is the IP address of a running Controller node, use the following command to disable fencing:

   ```
   $ ssh heat-admin@192.168.0.47 "sudo pcs property set stonith-enabled=false"
   ```

   Check the fencing status with the following command:

   ```
   $ ssh heat-admin@192.168.0.47 "sudo pcs property show stonith-enabled"
   ```

8. Check the **nova-compute** service on the director node:

   ```
   $ sudo systemctl status openstack-nova-compute
   $ openstack hypervisor list
   ```

   The output should show all non-maintenance mode nodes as **up**.

9. Make sure all undercloud services are running:

   ```
   $ sudo systemctl -t service
   ```

9.4.2. Removing a Ceph Monitor Daemon

This procedure removes a **ceph-mon** daemon from the storage cluster. If your Controller node is running a Ceph monitor service, complete the following steps to remove the ceph-mon daemon. This procedure assumes the Controller is reachable.

> **NOTE**
>
> A new Ceph monitor daemon will be added after a new Controller is added to the cluster.

1. Connect to the controller to be replaced and become root:

   ```
   # ssh heat-admin@192.168.0.47
   # sudo su -
   ```

   > **NOTE**
   >
   > If the controller is unreachable, skip steps 1 and 2 and continue the procedure at step 3 on any working controller node.

2. As root, stop the monitor:

   ```
   # systemctl stop ceph-mon@<monitor_hostname>
   ```

   For example:

   ```
   # systemctl stop ceph-mon@overcloud-controller-2
   ```

3. Remove the monitor from the cluster:

   ```
   # ceph mon remove <mon_id>
   ```

4. On the Ceph monitor node, remove the monitor entry from **/etc/ceph/ceph.conf**. For example, if you remove controller-2, then remove the IP and hostname for controller-2.
   Before:

   ```
   mon host = 172.18.0.21,172.18.0.22,172.18.0.24
   mon initial members = overcloud-controller-2,overcloud-controller-1,overcloud-controller-0
   ```

   After:

   ```
   mon host = 172.18.0.22,172.18.0.24
   mon initial members = overcloud-controller-1,overcloud-controller-0
   ```

5. Apply the same change to **/etc/ceph/ceph.conf** on the other overcloud nodes.

   NOTE

   The **ceph.conf** file is updated on the relevant overcloud nodes by director when the replacement controller node is added. Normally, this configuration file is managed only by director and should not be manually edited, but it is edited in this step to ensure consistency in case the other nodes restart before the new node is added.

6. Optionally, archive the monitor data and save it on another server:

   ```
   # mv /var/lib/ceph/mon/<cluster>-<daemon_id> /var/lib/ceph/mon/removed-<cluster>-<daemon_id>
   ```

9.4.3. Node Replacement

Identify the index of the node to remove. The node index is the suffix on the instance name from **nova list** output.

```
[stack@director ~]$ openstack server list
+--------------------------------------+------------------------+
| ID                                   | Name                   |
+--------------------------------------+------------------------+
| 861408be-4027-4f53-87a6-cd3cf206ba7a | overcloud-compute-0    |
| 0966e9ae-f553-447a-9929-c4232432f718 | overcloud-compute-1    |
| 9c08fa65-b38c-4b2e-bd47-33870bff06c7 | overcloud-compute-2    |
| a7f0f5e1-e7ce-4513-ad2b-81146bc8c5af | overcloud-controller-0 |
```

```
| cfefaf60-8311-4bc3-9416-6a824a40a9ae | overcloud-controller-1 |
| 97a055d4-aefd-481c-82b7-4a5f384036d2 | overcloud-controller-2 |
+--------------------------------------+------------------------+
```

In this example, the aim is to remove the **overcloud-controller-1** node and replace it with **overcloud-controller-3**. First, set the node into maintenance mode so the director does not reprovision the failed node. Correlate the instance ID from **nova list** with the node ID from **openstack baremetal node list**

```
[stack@director ~]$ openstack baremetal node list
+--------------------------------------+------+--------------------------------------+
| UUID                                 | Name | Instance UUID                        |
+--------------------------------------+------+--------------------------------------+
| 36404147-7c8a-41e6-8c72-a6e90afc7584 | None | 7bee57cf-4a58-4eaf-b851-2a8bf6620e48 |
| 91eb9ac5-7d52-453c-a017-c0e3d823efd0 | None | None                                 |
| 75b25e9a-948d-424a-9b3b-f0ef70a6eacf | None | None                                 |
| 038727da-6a5c-425f-bd45-fda2f4bd145b | None | 763bfec2-9354-466a-ae65-2401c13e07e5 |
| dc2292e6-4056-46e0-8848-d6e96df1f55d | None | 2017b481-706f-44e1-852a-2ee857c303c4 |
| c7eadcea-e377-4392-9fc3-cf2b02b7ec29 | None | 5f73c7d7-4826-49a5-b6be-8bfd558f3b41 |
| da3a8d19-8a59-4e9d-923a-6a336fe10284 | None | cfefaf60-8311-4bc3-9416-6a824a40a9ae |
| 807cb6ce-6b94-4cd1-9969-5c47560c2eee | None | c07c13e6-a845-4791-9628-260110829c3a |
+--------------------------------------+------+--------------------------------------+
```

Set the node into maintenance mode:

```
[stack@director ~]$ openstack baremetal node maintenance set da3a8d19-8a59-4e9d-923a-
6a336fe10284
```

Tag the new node with the **control** profile.

```
[stack@director ~]$ openstack baremetal node set --property
capabilities='profile:control,boot_option:local' 75b25e9a-948d-424a-9b3b-f0ef70a6eacf
```

The overcloud's database must continue running during the replacement procedure. To ensure Pacemaker does not stop Galera during this procedure, select a running Controller node and run the following command on the undercloud using the Controller node's IP address:

```
[stack@director ~]$ ssh heat-admin@192.168.0.47 "sudo pcs resource unmanage galera"
```

Create a YAML file (**~/templates/remove-controller.yaml**) that defines the node index to remove:

```
parameters:
  ControllerRemovalPolicies:
    [{'resource_list': ['1']}]
```

NOTE

You can speed up the replacement process by reducing the number for tries for settle in Corosync. Include the **CorosyncSettleTries** parameter in the **~/templates/remove-controller.yaml** environment file:

```
parameter_defaults:
  CorosyncSettleTries: 5
```

After identifying the node index, redeploy the overcloud and include the **remove-controller.yaml** environment file:

```
[stack@director ~]$ openstack overcloud deploy --templates --control-scale 3 -e ~/templates/remove-controller.yaml [OTHER OPTIONS]
```

If you passed any extra environment files or options when you created the overcloud, pass them again here to avoid making undesired changes to the overcloud.

However, note that the **-e ~/templates/remove-controller.yaml** is only required once in this instance.

The director removes the old node, creates a new one, and updates the overcloud stack. You can check the status of the overcloud stack with the following command:

```
[stack@director ~]$ openstack stack list --nested
```

9.4.4. Manual Intervention

During the **ControllerNodesPostDeployment** stage, the overcloud stack update halts with an **UPDATE_FAILED** error at **ControllerDeployment_Step1**. This is because some Puppet modules do not support nodes replacement. This point in the process requires some manual intervention. Follow these configuration steps:

1. Get a list of IP addresses for the Controller nodes. For example:

```
[stack@director ~]$ openstack server list
... +------------------------+ ... +-------------------------+
... | Name                   | ... | Networks                |
... +------------------------+ ... +-------------------------+
... | overcloud-compute-0    | ... | ctlplane=192.168.0.44   |
... | overcloud-controller-0 | ... | ctlplane=192.168.0.47   |
... | overcloud-controller-2 | ... | ctlplane=192.168.0.46   |
... | overcloud-controller-3 | ... | ctlplane=192.168.0.48   |
... +------------------------+ ... +-------------------------+
```

2. Check the **nodeid** value of the removed node in the **/etc/corosync/corosync.conf** file on an existing node. For example, the existing node is **overcloud-controller-0** at 192.168.0.47:

```
[stack@director ~]$ ssh heat-admin@192.168.0.47 "sudo cat /etc/corosync/corosync.conf"
```

This displays a **nodelist** that contains the ID for the removed node (**overcloud-controller-1**):

```
nodelist {
  node {
    ring0_addr: overcloud-controller-0
    nodeid: 1
  }
  node {
    ring0_addr: overcloud-controller-1
    nodeid: 2
  }
  node {
    ring0_addr: overcloud-controller-2
```

```
      nodeid: 3
    }
  }
```

Note the **nodeid** value of the removed node for later. In this example, it is 2.

3. Delete the failed node from the Corosync configuration on each node and restart Corosync. For this example, log into **overcloud-controller-0** and **overcloud-controller-2** and run the following commands:

```
[stack@director] ssh heat-admin@192.168.0.47 "sudo pcs cluster localnode remove
overcloud-controller-1"
[stack@director] ssh heat-admin@192.168.0.47 "sudo pcs cluster reload corosync"
[stack@director] ssh heat-admin@192.168.0.46 "sudo pcs cluster localnode remove
overcloud-controller-1"
[stack@director] ssh heat-admin@192.168.0.46 "sudo pcs cluster reload corosync"
```

4. Log into one of the remaining nodes and delete the node from the cluster with the **crm_node** command:

```
[stack@director] ssh heat-admin@192.168.0.47
[heat-admin@overcloud-controller-0 ~]$ sudo crm_node -R overcloud-controller-1 --force
```

Stay logged into this node.

5. Delete the failed node from the RabbitMQ cluster:

```
[heat-admin@overcloud-controller-0 ~]$ sudo rabbitmqctl forget_cluster_node
rabbit@overcloud-controller-1
```

6. Delete the failed node from MongoDB. First, find the IP address for the node's Interal API connection.

```
[heat-admin@overcloud-controller-0 ~]$ sudo netstat -tulnp | grep 27017
tcp        0      0 192.168.0.47:27017    0.0.0.0:*               LISTEN      13415/mongod
```

Check that the node is the **primary** replica set:

```
[root@overcloud-controller-0 ~]# echo "db.isMaster()" | mongo --host 192.168.0.47:27017
MongoDB shell version: 2.6.11
connecting to: 192.168.0.47:27017/echo
{
  "setName" : "tripleo",
  "setVersion" : 1,
  "ismaster" : true,
  "secondary" : false,
  "hosts" : [
    "192.168.0.47:27017",
    "192.168.0.46:27017",
    "192.168.0.45:27017"
  ],
  "primary" : "192.168.0.47:27017",
  "me" : "192.168.0.47:27017",
  "electionId" : ObjectId("575919933ea8637676159d28"),
    "maxBsonObjectSize" : 16777216,
    "maxMessageSizeBytes" : 48000000,
    "maxWriteBatchSize" : 1000,
    "localTime" : ISODate("2016-06-09T09:02:43.340Z"),
    "maxWireVersion" : 2,
    "minWireVersion" : 0,
    "ok" : 1
}
bye
```

This should indicate if the current node is the primary. If not, use the IP address of the node indicated in the **primary** key.

Connect to MongoDB on the primary node:

```
[heat-admin@overcloud-controller-0 ~]$ mongo --host 192.168.0.47
MongoDB shell version: 2.6.9
connecting to: 192.168.0.47:27017/test
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
http://docs.mongodb.org/
Questions? Try the support group
http://groups.google.com/group/mongodb-user
tripleo:PRIMARY>
```

Check the status of the MongoDB cluster:

```
tripleo:PRIMARY> rs.status()
```

Identify the node using the **_id** key and remove the failed node using the **name** key. In this case, we remove Node 1, which has **192.168.0.45:27017** for **name**:

```
tripleo:PRIMARY> rs.remove('192.168.0.45:27017')
```

IMPORTANT

You must run the command against the **PRIMARY** replica set. If you see the following message:

```
"replSetReconfig command must be sent to the current replica set primary."
```

Relog into MongoDB on the node designated as **PRIMARY**.

> **NOTE**
>
> The following output is normal when removing the failed node's replica set:
>
> ```
> 2016-05-07T03:57:19.541+0000 DBClientCursor::init call() failed
> 2016-05-07T03:57:19.543+0000 Error: error doing query: failed at
> src/mongo/shell/query.js:81
> 2016-05-07T03:57:19.545+0000 trying reconnect to 192.168.0.47:27017
> (192.168.0.47) failed
> 2016-05-07T03:57:19.547+0000 reconnect 192.168.0.47:27017
> (192.168.0.47) ok
> ```

Exit MongoDB:

```
tripleo:PRIMARY> exit
```

7. Update list of nodes in the Galera cluster:

```
[heat-admin@overcloud-controller-0 ~]$ sudo pcs resource update galera
wsrep_cluster_address=gcomm://overcloud-controller-0,overcloud-controller-3,overcloud-controller-2
```

8. Configure the Galera cluster check on the new node. Copy the **/etc/sysconfig/clustercheck** from the existing node to the same location on the new node.

9. Configure the **root** user's Galera access on the new node. Copy the **/root/.my.cnf** from the existing node to the same location on the new node.

10. Add the new node to the cluster:

```
[heat-admin@overcloud-controller-0 ~]$ sudo pcs cluster node add overcloud-controller-3
```

11. Check the **/etc/corosync/corosync.conf** file on each node. If the **nodeid** of the new node is the same as the removed node, update the value to a new nodeid value. For example, the **/etc/corosync/corosync.conf** file contains an entry for the new node (**overcloud-controller-3**):

```
nodelist {
  node {
    ring0_addr: overcloud-controller-0
    nodeid: 1
  }
  node {
    ring0_addr: overcloud-controller-2
    nodeid: 3
  }
  node {
    ring0_addr: overcloud-controller-3
    nodeid: 2
  }
}
```

Note that in this example, the new node uses the same **nodeid** of the removed node. Update this value to a unused node ID value. For example:

```
node {
  ring0_addr: overcloud-controller-3
  nodeid: 4
}
```

Update this **nodeid** value on each Controller node’s **/etc/corosync/corosync.conf** file, including the new node.

12. Restart the Corosync service on the existing nodes only. For example, on **overcloud-controller-0**:

    ```
    [heat-admin@overcloud-controller-0 ~]$ sudo pcs cluster reload corosync
    ```

    And on **overcloud-controller-2**:

    ```
    [heat-admin@overcloud-controller-2 ~]$ sudo pcs cluster reload corosync
    ```

    Do not run this command on the new node.

13. Start the new Controller node:

    ```
    [heat-admin@overcloud-controller-0 ~]$ sudo pcs cluster start overcloud-controller-3
    ```

14. Restart the Galera cluster and return it to Pacemaker management:

    ```
    [heat-admin@overcloud-controller-0 ~]$ sudo pcs resource cleanup galera
    [heat-admin@overcloud-controller-0 ~]$ sudo pcs resource manage galera
    ```

15. Enable and restart some services through Pacemaker. The cluster is currently in maintenance mode and you will need to temporarily disable it to enable the service. For example:

    ```
    [heat-admin@overcloud-controller-3 ~]$ sudo pcs property set maintenance-mode=false --
    wait
    ```

16. Wait until the Galera service starts on all nodes.

    ```
    [heat-admin@overcloud-controller-3 ~]$ sudo pcs status | grep galera -A1
    Master/Slave Set: galera-master [galera]
    Masters: [ overcloud-controller-0 overcloud-controller-2 overcloud-controller-3 ]
    ```

    If need be, perform a **cleanup** on the new node:

    ```
    [heat-admin@overcloud-controller-3 ~]$ sudo pcs resource cleanup galera --node overcloud-
    controller-3
    ```

17. Switch the cluster back into maintenance mode:

    ```
    [heat-admin@overcloud-controller-3 ~]$ sudo pcs property set maintenance-mode=true --
    wait
    ```

The manual configuration is complete. Re-run the overcloud deployment command to continue the stack update:

```
[stack@director ~]$ openstack overcloud deploy --templates --control-scale 3 [OTHER OPTIONS]
```

IMPORTANT

If you passed any extra environment files or options when you created the overcloud, pass them again here to avoid making undesired changes to the overcloud. However, note that the **remove-controller.yaml** file is no longer needed.

9.4.5. Finalizing Overcloud Services

After the overcloud stack update completes, some final configuration is required. Log in to one of the Controller nodes and refresh any stopped services in Pacemaker:

```
[heat-admin@overcloud-controller-0 ~]$ for i in `sudo pcs status|grep -B2 Stop |grep -v
"Stop\|Start"|awk -F"[" '/\[/ {print substr($NF,0,length($NF)-1)}'`; do echo $i; sudo pcs resource
cleanup $i; done
```

Perform a final status check to make sure services are running correctly:

```
[heat-admin@overcloud-controller-0 ~]$ sudo pcs status
```

NOTE

If any services have failed, use the **pcs resource cleanup** command to restart them after resolving them.

If the Controller nodes use fencing, delete the old fencing record and create a new one:

```
[heat-admin@overcloud-controller-0 ~]$ sudo pcs stonith show
[heat-admin@overcloud-controller-0 ~]$ sudo pcs stonish delete my-ipmilan-for-controller-1
[heat-admin@overcloud-controller-0 ~]$ sudo pcs stonith create my-ipmilan-for-controller-3
fence_ipmilan pcmk_host_list=overcloud-controller-3 ipaddr=192.0.2.208 login=admin
passwd=p@55w0rd! lanplus=1 cipher=1 op monitor interval=60s
[heat-admin@overcloud-controller-0 ~]$ sudo pcs constraint location my-ipmilan-for-controller-3
avoids overcloud-controller-3
```

Re-enable fencing:

```
[heat-admin@overcloud-controller-0 ~]$ sudo pcs property set stonith-enabled=true
```

NOTE

For more information on fencing configuration, see Section 7.7, “Fencing the Controller Nodes”.

Exit to the director

```
[heat-admin@overcloud-controller-0 ~]$ exit
```

9.4.6. Finalizing L3 Agent Router Hosting

Source the **overcloudrc** file so that you can interact with the overcloud. Check your routers to make sure the L3 agents are properly hosting the routers in your overcloud environment. In this example, we use a router with the name **r1**:

```
[stack@director ~]$ source ~/overcloudrc
[stack@director ~]$ neutron l3-agent-list-hosting-router r1
```

This list might still show the old node instead of the new node. To replace it, list the L3 network agents in your environment:

```
[stack@director ~]$ neutron agent-list | grep "neutron-l3-agent"
```

Identify the UUID for the agents on the new node and the old node. Add the router to the agent on the new node and remove the router from old node. For example:

```
[stack@director ~]$ neutron l3-agent-router-add fd6b3d6e-7d8c-4e1a-831a-4ec1c9ebb965 r1
[stack@director ~]$ neutron l3-agent-router-remove b40020af-c6dd-4f7a-b426-eba7bac9dbc2 r1
```

Perform a final check on the router and make all are active:

```
[stack@director ~]$ neutron l3-agent-list-hosting-router r1
```

Delete the existing Neutron agents that point to old Controller node. For example:

```
[stack@director ~]$ neutron agent-list -F id -F host | grep overcloud-controller-1
| ddae8e46-3e8e-4a1b-a8b3-c87f13c294eb | overcloud-controller-1.localdomain |
[stack@director ~]$ neutron agent-delete ddae8e46-3e8e-4a1b-a8b3-c87f13c294eb
```

9.4.7. Finalizing Compute Services

Compute services for the removed node still exist in the overcloud and require removal. Source the **overcloudrc** file so that you can interact with the overcloud. Check the compute services for the removed node:

```
[stack@director ~]$ source ~/overcloudrc
[stack@director ~]$ nova service-list | grep "overcloud-controller-1.localdomain"
```

Remove the compute services for the node. For example, if the **nova-scheduler** service for **overcloud-controller-1.localdomain** has an ID of 5, run the following command:

```
[stack@director ~]$ nova service-delete 5
```

Perform this task for each service of the removed node.

Check the **openstack-nova-consoleauth** service on the new node.

```
[stack@director ~]$ nova service-list | grep consoleauth
```

If the service is not running, log into a Controller node and restart the service:

```
[stack@director] ssh heat-admin@192.168.0.47
[heat-admin@overcloud-controller-0 ~]$ pcs resource restart openstack-nova-consoleauth
```

9.4.8. Conclusion

The failed Controller node and its related services are now replaced with a new node.

IMPORTANT

If you disabled automatic ring building for Object Storage, like in Section 9.6, “Replacing Object Storage Nodes”, you need to manually build the Object Storage ring files for the new node. See Section 9.6, “Replacing Object Storage Nodes” for more information on manually building ring files.

9.5. REPLACING CEPH STORAGE NODES

The director provides a method to replace Ceph Storage nodes in a director-created cluster. You can find these instructions in the Red Hat Ceph Storage for the Overcloud .

9.6. REPLACING OBJECT STORAGE NODES

This section describes how to replace Object Storage nodes while maintaining the integrity of the cluster. In this example, we have a two-node Object Storage cluster where the node **overcloud-objectstorage-1** needs to be replaced. Our aim is to add one more node, then remove **overcloud-objectstorage-1** (effectively replacing it).

1. Create an environment file called ~/templates/swift-upscale.yaml with the following content:

   ```
   parameter_defaults:
     ObjectStorageCount: 3
   ```

   The **ObjectStorageCount** defines how many Object Storage nodes in our environment. In this situation, we scale from 2 to 3 nodes.

2. Include the **swift-upscale.yaml** file with the rest of your overcloud’s environment files (*ENVIRONMENT_FILES*) as part of the **openstack overcloud deploy**:

   ```
   $ openstack overcloud deploy --templates ENVIRONMENT_FILES -e swift-upscale.yaml
   ```

   NOTE

   Add **swift-upscale.yaml** to the end of the environment file list so its parameters supersede previous environment file parameters.

   After redeployment completes, the overcloud now contains an additional Object Storage node.

3. Data now needs to be replicated to the new node. Before removing a node (in this case, **overcloud-objectstorage-1**) you should wait for a *replication pass* to finish on the new node. You can check the replication pass progress in **/var/log/swift/swift.log**. When the pass finishes, the Object Storage service should log entries similar to the following:

   ```
   Mar 29 08:49:05 localhost object-server: Object replication complete.
   Mar 29 08:49:11 localhost container-server: Replication run OVER
   Mar 29 08:49:13 localhost account-server: Replication run OVER
   ```

4. To remove the old node from the ring, reduce the **ObjectStorageCount** in **swift-upscale.yaml** to the omit the old ring. In this case, we reduce it to 2:

   ```
   parameter_defaults:
     ObjectStorageCount: 2
   ```

5. Create a new environment file named **remove-object-node.yaml**. This file will identify and remove the specified Object Storage node. The following content specifies the removal of **overcloud-objectstorage-1**:

   ```
   parameter_defaults:
     ObjectStorageRemovalPolicies:
       [{'resource_list': ['1']}]
   ```

6. Include both environment files with the deployment command:

   ```
   $ openstack overcloud deploy --templates ENVIRONMENT_FILES -e swift-upscale.yaml -e
   remove-object-node.yaml ...
   ```

The director deletes the Object Storage node from the overcloud and updates the rest of the nodes on the overcloud to accommodate the node removal.

CHAPTER 10. REBOOTING NODES

Some situations require a reboot of nodes in the undercloud and overcloud. The following procedures show how to reboot different node types. Be aware of the following notes:

- If rebooting all nodes in one role, it is advisable to reboot each node individually. This helps retain services for that role during the reboot.
- If rebooting all nodes in your OpenStack Platform environment, use the following list to guide the reboot order:

Recommended Node Reboot Order

1. Reboot the director
2. Reboot Controller nodes
3. Reboot standalone Ceph MON nodes
4. Reboot Ceph Storage nodes
5. Reboot Compute nodes
6. Reboot object Storage nodes

10.1. REBOOTING THE DIRECTOR

To reboot the director node, follow this process:

1. Reboot the node:

```
$ sudo reboot
```

2. Wait until the node boots.
3. When the node boots, check the status of all services:

```
$ sudo systemctl list-units "openstack*" "neutron*" "openvswitch*"
```

> **NOTE**
>
> It might take approximately 10 minutes for the **openstack-nova-compute** to become active after a reboot.

4. Verify the existence of your Overcloud and its nodes:

```
$ source ~/stackrc
$ openstack server list
$ openstack baremetal node list
$ openstack stack list
```

10.2. REBOOTING CONTROLLER NODES

To reboot the Controller nodes, follow this process:

1. Select a node to reboot. Log into it and stop the cluster before rebooting:

   ```
   $ sudo pcs cluster stop
   ```

2. Reboot the cluster:

   ```
   $ sudo reboot
   ```

   The remaining Controller Nodes in the cluster retain the high availability services during the reboot.

3. Wait until the node boots.

4. Re-enable the cluster for the node:

   ```
   $ sudo pcs cluster start
   ```

5. Log into the node and check the cluster status:

   ```
   $ sudo pcs status
   ```

   The node rejoins the cluster.

   > NOTE
   >
   > If any services fail after the reboot, run sudo **pcs resource cleanup**, which cleans the errors and sets the state of each resource to **Started**. If any errors persist, contact Red Hat and request guidance and assistance.

6. Check all **systemd** services on the Controller Node are active:

   ```
   $ sudo systemctl list-units "openstack*" "neutron*" "openvswitch*"
   ```

7. Log out of the node, select the next Controller Node to reboot, and repeat this procedure until you have rebooted all Controller Nodes.

10.3. REBOOTING STANDALONE CEPH MON NODES

To reboot the Ceph MON nodes, follow this process:

1. Log into 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.

10.4. REBOOTING CEPH STORAGE NODES

To reboot the Ceph Storage nodes, follow this process:

1. Log into a Ceph MON or Controller node and disable Ceph Storage cluster rebalancing temporarily:

   ```
   $ sudo ceph osd set noout
   $ sudo ceph osd set norebalance
   ```

2. Select the first Ceph Storage node to reboot and log into it.

3. Reboot the node:

   ```
   $ sudo reboot
   ```

4. Wait until the node boots.

5. Log into the node and check the cluster status:

   ```
   $ sudo ceph -s
   ```

   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 into a Ceph MON or Controller node and enable cluster rebalancing again:

   ```
   $ sudo ceph osd unset noout
   $ sudo ceph osd unset norebalance
   ```

8. Perform a final status check to verify the cluster reports **HEALTH_OK**:

   ```
   $ sudo ceph status
   ```

10.5. REBOOTING COMPUTE NODES

Reboot each Compute node individually and ensure zero downtime of instances in your OpenStack Platform environment. This involves the following workflow:

1. Select a Compute node to reboot

2. Migrate its instances to another Compute node

3. Reboot the empty Compute node

List all Compute nodes and their UUIDs:

```
$ nova list | grep "compute"
```

Select a Compute node to reboot and first migrate its instances using the following process:

1. From the undercloud, select a Compute Node to reboot and disable it:

```
$ source ~/overcloudrc
$ openstack compute service list
$ openstack compute service set [hostname] nova-compute --disable
```

2. List all instances on the Compute node:

```
$ openstack server list --host [hostname] --all-projects
```

3. Migrate each instance from the disabled host. Use one of the following commands:

   a. Migrate the instance to a specific host of your choice:

   ```
   $ openstack server migrate [instance-id] --live [target-host]--wait
   ```

   b. Let **nova-scheduler** automatically select the target host:

   ```
   $ nova live-migration [instance-id]
   ```

   NOTE

   The **nova** command might cause some deprecation warnings, which are safe to ignore.

4. Wait until migration completes.

5. Confirm the instance has migrated from the Compute node:

```
$ openstack server list --host [hostname] --all-projects
```

6. Repeat this step until you have migrated all instances from the Compute Node.

IMPORTANT

For full instructions on configuring and migrating instances, see Chapter 8, *Migrating Virtual Machines Between Compute Nodes*.

Reboot the Compute node using the following process

1. Log into the Compute Node and reboot it:

```
$ sudo reboot
```

2. Wait until the node boots.

3. Enable the Compute Node again:

```
$ source ~/overcloudrc
$ openstack compute service set [hostname] nova-compute --enable
```

4. Select the next node to reboot.

10.6. REBOOTING OBJECT STORAGE NODES

To reboot the Object Storage nodes, follow this process:

1. Select a Object Storage node to reboot. Log into it and reboot it:

   ```
   $ sudo reboot
   ```

2. Wait until the node boots.

3. Log into the node and check the status:

   ```
   $ sudo systemctl list-units "openstack-swift*"
   ```

4. Log out of the node and repeat this process on the next Object Storage node.

CHAPTER 11. TROUBLESHOOTING DIRECTOR ISSUES

An error can occur at certain stages of the director's processes. This section provides some information for diagnosing common problems.

Note the common logs for the director's components:

- The **/var/log** directory contains logs for many common OpenStack Platform components as well as logs for standard Red Hat Enterprise Linux applications.
- The **journald** service provides logs for various components. Note that ironic uses two units: **openstack-ironic-api** and **openstack-ironic-conductor**. Likewise, **ironic-inspector** uses two units as well: **openstack-ironic-inspector** and **openstack-ironic-inspector-dnsmasq**. Use both units for each respective component. For example:

  ```
  $ sudo journalctl -u openstack-ironic-inspector -u openstack-ironic-inspector-dnsmasq
  ```

- **ironic-inspector** also stores the ramdisk logs in **/var/log/ironic-inspector/ramdisk/** as gz-compressed tar files. Filenames contain date, time, and the IPMI address of the node. Use these logs for diagnosing introspection issues.

11.1. TROUBLESHOOTING NODE REGISTRATION

Issues with node registration usually arise from issues with incorrect node details. In this case, use **ironic** to fix problems with node data registered. Here are a few examples:

Find out the assigned port UUID:

```
$ ironic node-port-list [NODE UUID]
```

Update the MAC address:

```
$ ironic port-update [PORT UUID] replace address=[NEW MAC]
```

Run the following command:

```
$ ironic node-update [NODE UUID] replace driver_info/ipmi_address=[NEW IPMI ADDRESS]
```

11.2. TROUBLESHOOTING HARDWARE INTROSPECTION

The introspection process must run to completion. However, ironic's Discovery daemon (**ironic-inspector**) times out after a default 1 hour period if the discovery ramdisk provides no response. Sometimes this might indicate a bug in the discovery ramdisk but usually it happens due to an environment misconfiguration, particularly BIOS boot settings.

Here are some common scenarios where environment misconfiguration occurs and advice on how to diagnose and resolve them.

Errors with Starting Node Introspection

Normally the introspection process uses the **baremetal introspection**, which acts an an umbrella command for ironic's services. However, if running the introspection directly with **ironic-inspector**, it might fail to discover nodes in the AVAILABLE state, which is meant for deployment and not for discovery. Change the node status to the MANAGEABLE state before discovery:

```
$ ironic node-set-provision-state [NODE UUID] manage
```

Then, when discovery completes, change back to AVAILABLE before provisioning:

```
$ ironic node-set-provision-state [NODE UUID] provide
```

Introspected node is not booting in PXE

Before a node reboots, **ironic-inspector** adds the MAC address of the node to the undercloud firewall's **ironic-inspector** chain. This allows the node to boot over PXE. To verify the correct configuration, run the following command:

```
$ `sudo iptables -L`
```

The output should display the following chain table with the MAC address:

```
Chain ironic-inspector (1 references)
target     prot opt source               destination
DROP       all  --  anywhere             anywhere             MAC xx:xx:xx:xx:xx:xx
ACCEPT     all  --  anywhere             anywhere
```

If the MAC address is not there, the most common cause is a corruption in the **ironic-inspector** cache, which is in an SQLite database. To fix it, delete the SQLite file:

```
$ sudo rm /var/lib/ironic-inspector/inspector.sqlite
```

And recreate it:

```
$ sudo ironic-inspector-dbsync --config-file /etc/ironic-inspector/inspector.conf upgrade
$ sudo systemctl restart openstack-ironic-inspector
```

Stopping the Discovery Process

Currently **ironic-inspector** does not provide a direct means for stopping discovery. The recommended path is to wait until the process times out. If necessary, change the **timeout** setting in **/etc/ironic-inspector/inspector.conf** to change the timeout period to another period in minutes.

In worst case scenarios, you can stop discovery for all nodes using the following process:

Change the power state of each node to off:

```
$ ironic node-set-power-state [NODE UUID] off
```

Remove **ironic-inspector** cache and restart it:

```
$ rm /var/lib/ironic-inspector/inspector.sqlite
```

Resynchronize the **ironic-inspector** cache:

```
$ sudo ironic-inspector-dbsync --config-file /etc/ironic-inspector/inspector.conf upgrade
$ sudo systemctl restart openstack-ironic-inspector
```

Accessing the Introspection Ramdisk

The introspection ramdisk uses a dynamic login element. This means you can provide either a temporary password or an SSH key to access the node during introspection debugging. Use the following process to set up ramdisk access:

1. Provide a temporary password to the **openssl passwd -1** command to generate an MD5 hash. For example:

```
$ openssl passwd -1 mytestpassword
$1$enjRSyIw$/fYUpJwr6abFy/d.koRgQ/
```

2. Edit the **/httpboot/inspector.ipxe** file, find the line starting with **kernel**, and append the **rootpwd** parameter and the MD5 hash. For example:

```
kernel http://192.2.0.1:8088/agent.kernel ipa-inspection-callback-
url=http://192.168.0.1:5050/v1/continue ipa-inspection-collectors=default,extra-hardware,logs
systemd.journald.forward_to_console=yes BOOTIF=${mac} ipa-debug=1 ipa-inspection-
benchmarks=cpu,mem,disk rootpwd="$1$enjRSyIw$/fYUpJwr6abFy/d.koRgQ/" selinux=0
```

Alternatively, you can append the **sshkey** parameter with your public SSH key.

NOTE

Quotation marks are required for both the **rootpwd** and **sshkey** parameters.

3. Start the introspection and find the IP address from either the **arp** command or the DHCP logs:

```
$ arp
$ sudo journalctl -u openstack-ironic-inspector-dnsmasq
```

4. SSH as a root user with the temporary password or the SSH key.

```
$ ssh root@192.0.2.105
```

Checking Introspection Storage

The director uses OpenStack Object Storage (swift) to save the hardware data obtained during the introspection process. If this service is not running, the introspection can fail. Check all services related to OpenStack Object Storage to ensure the service is running:

```
$ sudo systemctl list-units openstack-swift*
```

11.3. TROUBLESHOOTING WORKFLOWS AND EXECUTIONS

The OpenStack Workflow (mistral) service groups multiple OpenStack tasks into workflows. Red Hat OpenStack Platform uses a set of these workflow to perform common functions across the CLI and web UI. This includes bare metal node control, validations, plan management, and overcloud deployment.

For example, when running the **openstack overcloud deploy** command, the OpenStack Workflow service executes two workflows. The first one uploads the deployment plan:

```
Removing the current plan files
```

```
Uploading new plan files
Started Mistral Workflow. Execution ID: aef1e8c6-a862-42de-8bce-073744ed5e6b
Plan updated
```

The second one starts the overcloud deployment:

```
Deploying templates in the directory /tmp/tripleoclient-LhRlHX/tripleo-heat-templates
Started Mistral Workflow. Execution ID: 97b64abe-d8fc-414a-837a-1380631c764d
2016-11-28 06:29:26Z [overcloud]: CREATE_IN_PROGRESS  Stack CREATE started
2016-11-28 06:29:26Z [overcloud.Networks]: CREATE_IN_PROGRESS  state changed
2016-11-28 06:29:26Z [overcloud.HeatAuthEncryptionKey]: CREATE_IN_PROGRESS  state
changed
2016-11-28 06:29:26Z [overcloud.ServiceNetMap]: CREATE_IN_PROGRESS  state changed
...
```

Workflow Objects

OpenStack Workflow uses the following objects to keep track of the workflow:

Actions

A particular instruction that OpenStack performs once an associated task runs. Examples include running shell scripts or performing HTTP requests. Some OpenStack components have in-built actions that OpenStack Workflow uses.

Tasks

Defines the action to run and the result of running the action. These tasks usually have actions or other workflows associated with them. Once a task completes, the workflow directs to another task, usually depending on whether the task succeeded or failed.

Workflows

A set of tasks grouped together and executed in a specific order.

Executions

Defines a particular action, task, or workflow running.

Workflow Error Diagnosis

OpenStack Workflow also provides robust logging of executions, which help you identify issues with certain command failures. For example, if a workflow execution fails, you can identify the point of failure. List the workflow executions that have the failed state **ERROR**:

```
$ mistral execution-list | grep "ERROR"
```

Get the UUID of the failed workflow execution (for example, 3c87a885-0d37-4af8-a471-1b392264a7f5) and view the execution and its output:

```
$ mistral execution-get 3c87a885-0d37-4af8-a471-1b392264a7f5
$ mistral execution-get-output 3c87a885-0d37-4af8-a471-1b392264a7f5
```

This provides information about the failed task in the execution. The **mistral execution-get** also displays the workflow used for the execution (for example, **tripleo.plan_management.v1.update_deployment_plan**). You can view the full workflow definition using the following command:

```
$ mistral execution-get-definition tripleo.plan_management.v1.update_deployment_plan
```

This is useful for identifying where in the workflow a particular task occurs.

You can also view action executions and their results using a similar command syntax:

```
$ mistral action-execution-list
$ mistral action-execution-get b59245bf-7183-4fcf-9508-c83ec1a26908
$ mistral action-execution-get-output b59245bf-7183-4fcf-9508-c83ec1a26908
```

This is useful for identifying a specific action causing issues.

11.4. TROUBLESHOOTING OVERCLOUD CREATION

There are three layers where the deployment can fail:

- Orchestration (heat and nova services)
- Bare Metal Provisioning (ironic service)
- Post-Deployment Configuration (Puppet)

If an overcloud deployment has failed at any of these levels, use the OpenStack clients and service log files to diagnose the failed deployment.

11.4.1. Orchestration

In most cases, Heat shows the failed overcloud stack after the overcloud creation fails:

```
$ heat stack-list
+-----------------------+------------+--------------------+----------------------+
| id                    | stack_name | stack_status       | creation_time        |
+-----------------------+------------+--------------------+----------------------+
| 7e88af95-535c-4a55... | overcloud  | CREATE_FAILED      | 2015-04-06T17:57:16Z |
+-----------------------+------------+--------------------+----------------------+
```

If the stack list is empty, this indicates an issue with the initial Heat setup. Check your Heat templates and configuration options, and check for any error messages that presented after running **openstack overcloud deploy**.

11.4.2. Bare Metal Provisioning

Check **ironic** to see all registered nodes and their current status:

```
$ ironic 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       |
+----------+------+---------------+-------------+-----------------+-------------+
```

Here are some common issues that arise from the provisioning process.

- Review the Provision State and Maintenance columns in the resulting table. Check for the following:
  - An empty table, or fewer nodes than you expect
  - Maintenance is set to True
  - Provision State is set to **manageable**. This usually indicates an issue with the registration or discovery processes. For example, if Maintenance sets itself to True automatically, the nodes are usually using the wrong power management credentials.
- If Provision State is **available**, then the problem occurred before bare metal deployment has even started.
- If Provision State is **active** and Power State is **power on**, the bare metal deployment has finished successfully. This means that the problem occurred during the post-deployment configuration step.
- If Provision State is **wait call-back** for a node, the bare metal provisioning process has not yet finished for this node. Wait until this status changes, otherwise, connect to the virtual console of the failed node and check the output.
- If Provision State is **error** or **deploy failed**, then bare metal provisioning has failed for this node. Check the bare metal node's details:

  ```
  $ ironic node-show [NODE UUID]
  ```

  Look for **last_error** field, which contains error description. If the error message is vague, you can use logs to clarify it:

  ```
  $ sudo journalctl -u openstack-ironic-conductor -u openstack-ironic-api
  ```
- If you see **wait timeout error** and the node Power State is **power on**, connect to the virtual console of the failed node and check the output.

11.4.3. Post-Deployment Configuration

Many things can occur during the configuration stage. For example, a particular Puppet module could fail to complete due to an issue with the setup. This section provides a process to diagnose such issues.

List all the resources from the overcloud stack to see which one failed:

```
$ heat resource-list overcloud
```

This shows a table of all resources and their states. Look for any resources with a **CREATE_FAILED**.

Show the failed resource:

```
$ heat resource-show overcloud [FAILED RESOURCE]
```

Check for any information in the **resource_status_reason** field that can help your diagnosis.

Use the **nova** command to see the IP addresses of the overcloud nodes.

```
$ nova list
```

Log in as the **heat-admin** user to one of the deployed nodes. For example, if the stack's resource list shows the error occurred on a Controller node, log in to a Controller node. The **heat-admin** user has sudo access.

```
$ ssh heat-admin@192.0.2.14
```

Check the **os-collect-config** log for a possible reason for the failure.

```
$ sudo journalctl -u os-collect-config
```

In some cases, nova fails deploying the node in entirety. This situation would be indicated by a failed **OS::Heat::ResourceGroup** for one of the overcloud role types. Use **nova** to see the failure in this case.

```
$ nova list
$ nova show [SERVER ID]
```

The most common error shown will reference the error message **No valid host was found**. See Section 11.6, "Troubleshooting "No Valid Host Found" Errors" for details on troubleshooting this error. In other cases, look at the following log files for further troubleshooting:

- **/var/log/nova/***
- **/var/log/heat/***
- **/var/log/ironic/***

Use the SOS toolset, which gathers information about system hardware and configuration. Use this information for diagnostic purposes and debugging. SOS is commonly used to help support technicians and developers. SOS is useful on both the undercloud and overcloud. Install the **sos** package:

```
$ sudo yum install sos
```

Generate a report:

```
$ sudo sosreport --all-logs
```

The post-deployment process for Controller nodes uses five main steps for the deployment. This includes:

Table 11.1. Controller Node Configuration Steps

Step	Description
ControllerDeployment_Step1	Initial load balancing software configuration, including Pacemaker, RabbitMQ, Memcached, Redis, and Galera.
ControllerDeployment_Step2	Initial cluster configuration, including Pacemaker configuration, HAProxy, MongoDB, Galera, Ceph Monitor, and database initialization for OpenStack Platform services.

ControllerDeployment_Step3	Initial ring build for OpenStack Object Storage (**swift**). Configuration of all OpenStack Platform services (**nova**, **neutron**, **cinder**, **sahara**, **ceilometer**, **heat**, **horizon**, **aodh**, **gnocchi**).
ControllerDeployment_Step4	Configure service start up settings in Pacemaker, including constraints to determine service start up order and service start up parameters.
ControllerDeployment_Step5	Initial configuration of projects, roles, and users in OpenStack Identity (**keystone**).

11.5. TROUBLESHOOTING IP ADDRESS CONFLICTS ON THE PROVISIONING NETWORK

Discovery and deployment tasks will fail if the destination hosts are allocated an IP address which is already in use. To avoid this issue, you can perform a port scan of the Provisioning network to determine whether the discovery IP range and host IP range are free.

Perform the following steps from the undercloud host:

Install **nmap**:

```
# yum install nmap
```

Use **nmap** to scan the IP address range for active addresses. This example scans the 192.0.2.0/24 range, replace this with the IP subnet of the Provisioning network (using CIDR bitmask notation):

```
# nmap -sn 192.0.2.0/24
```

Review the output of the **nmap** scan:

For example, you should see the IP address(es) of the undercloud, and any other hosts that are present on the subnet. If any of the active IP addresses conflict with the IP ranges in undercloud.conf, you will need to either change the IP address ranges or free up the IP addresses before introspecting or deploying the overcloud nodes.

```
# nmap -sn 192.0.2.0/24

Starting Nmap 6.40 ( http://nmap.org ) at 2015-10-02 15:14 EDT
Nmap scan report for 192.0.2.1
Host is up (0.00057s latency).
Nmap scan report for 192.0.2.2
Host is up (0.00048s latency).
Nmap scan report for 192.0.2.3
Host is up (0.00045s latency).
Nmap scan report for 192.0.2.5
Host is up (0.00040s latency).
Nmap scan report for 192.0.2.9
Host is up (0.00019s latency).
Nmap done: 256 IP addresses (5 hosts up) scanned in 2.45 seconds
```

11.6. 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 means the nova Scheduler could not find a bare metal node suitable for booting the new instance. This in turn usually means a mismatch between resources that nova expects to find and resources that ironic advertised to nova. Check the following in this case:

1. Make sure introspection succeeds for you. Otherwise check that each node contains the required ironic node properties. For each node:

   ```
   $ ironic node-show [NODE UUID]
   ```

   Check the **properties** JSON field has valid values for keys **cpus**, **cpu_arch**, **memory_mb** and **local_gb**.

2. Check that the nova flavor used does not exceed the ironic node properties above for a required number of nodes:

   ```
   $ nova flavor-show [FLAVOR NAME]
   ```

3. Check that sufficient nodes are in the **available** state according to **ironic node-list**. Nodes in **manageable** state usually mean a failed introspection.

4. Check the nodes are not in maintenance mode. Use **ironic node-list** to check. A node automatically changing to maintenance mode usually means incorrect power credentials. Check them and then remove maintenance mode:

   ```
   $ ironic node-set-maintenance [NODE UUID] off
   ```

5. If you’re using the Automated Health Check (AHC) tools to perform automatic node tagging, check that you have enough nodes corresponding to each flavor/profile. Check the **capabilities** key in **properties** field for **ironic node-show**. For example, a node tagged for the Compute role should contain **profile:compute**.

6. It takes some time for node information to propagate from ironic to nova after introspection. The director’s tool usually accounts for it. However, if you performed some steps manually, there might be a short period of time when nodes are not available to nova. Use the following command to check the total resources in your system.:

   ```
   $ nova hypervisor-stats
   ```

11.7. TROUBLESHOOTING THE OVERCLOUD AFTER CREATION

After creating your overcloud, you might want to perform certain overcloud operations in the future. For example, you might aim to scale your available nodes, or replace faulty nodes. Certain issues might arise when performing these operations. This section provides some advice to diagnose and troubleshoot failed post-creation operations.

11.7.1. Overcloud Stack Modifications

Problems can occur when modifying the **overcloud** stack through the director. Example of stack modifications include:

- Scaling Nodes
- Removing Nodes
- Replacing Nodes

Modifying the stack is similar to the process of creating the stack, in that the director checks the availability of the requested number of nodes, provisions additional or removes existing nodes, and then applies the Puppet configuration. Here are some guidelines to follow in situations when modifying the **overcloud** stack.

As an initial step, follow the advice set in Section 11.4.3, "Post-Deployment Configuration". These same steps can help diagnose problems with updating the **overcloud** heat stack. In particular, use the following command to help identify problematic resources:

heat stack-list --show-nested

List all stacks. The **--show-nested** displays all child stacks and their respective parent stacks. This command helps identify the point where a stack failed.

heat resource-list overcloud

List all resources in the **overcloud** stack and their current states. This helps identify which resource is causing failures in the stack. You can trace this resource failure to its respective parameters and configuration in the heat template collection and the Puppet modules.

heat event-list overcloud

List all events related to the **overcloud** stack in chronological order. This includes the initiation, completion, and failure of all resources in the stack. This helps identify points of resource failure.

The next few sections provide advice to diagnose issues on specific node types.

11.7.2. Controller Service Failures

The overcloud Controller nodes contain the bulk of Red Hat OpenStack Platform services. Likewise, you might use multiple Controller nodes in a high availability cluster. If a certain service on a node is faulty, the high availability cluster provides a certain level of failover. However, it then becomes necessary to diagnose the faulty service to ensure your overcloud operates at full capacity.

The Controller nodes use Pacemaker to manage the resources and services in the high availability cluster. The Pacemaker Configuration System (**pcs**) command is a tool that manages a Pacemaker cluster. Run this command on a Controller node in the cluster to perform configuration and monitoring functions. Here are few commands to help troubleshoot overcloud services on a high availability cluster:

pcs status

Provides a status overview of the entire cluster including enabled resources, failed resources, and online nodes.

pcs resource show

Shows a list of resources, and their respective nodes.

pcs resource disable [resource]

Stop a particular resource.

pcs resource enable [resource]

Start a particular resource.

pcs cluster standby [node]

Place a node in standby mode. The node is no longer available in the cluster. This is useful for performing maintenance on a specific node without affecting the cluster.

pcs cluster unstandby [node]

Remove a node from standby mode. The node becomes available in the cluster again.

Use these Pacemaker commands to identify the faulty component and/or node. After identifying the component, view the respective component log file in **/var/log/**.

11.7.3. Compute Service Failures

Compute nodes use the Compute service to perform hypervisor-based operations. This means the main diagnosis for Compute nodes revolves around this service. For example:

- View the status of the service using the following **systemd** function:

  ```
  $ sudo systemctl status openstack-nova-compute.service
  ```

  Likewise, view the **systemd** journal for the service using the following command:

  ```
  $ sudo journalctl -u openstack-nova-compute.service
  ```

- The primary log file for Compute nodes is **/var/log/nova/nova-compute.log**. If issues occur with Compute node communication, this log file is usually a good place to start a diagnosis.

- If performing maintenance on the Compute node, migrate the existing instances from the host to an operational Compute node, then disable the node. See Chapter 8, *Migrating Virtual Machines Between Compute Nodes* for more information on node migrations.

11.7.4. Ceph Storage Service Failures

For any issues that occur with Red Hat Ceph Storage clusters, see Chapter 10. Logging and Debugging in the Red Hat Ceph Storage Configuration Guide. This section provides information on diagnosing logs for all Ceph storage services.

11.8. TUNING THE UNDERCLOUD

The advice in this section aims to help increase the performance of your undercloud. Implement the recommendations as necessary.

- The Identity Service (keystone) uses a token-based system for access control against the other OpenStack services. After a certain period, the database will accumulate a large number of unused tokens; a default cronjob flushes the token table every day. It is recommended that you monitor your environment and adjust the token flush interval as needed. For the undercloud, you can adjust the interval using **crontab -u keystone -e**. Note that this is a temporary change and that **openstack undercloud update** will reset this cronjob back to its default.

- Heat stores a copy of all template files in its database's **raw_template** table each time you run **openstack overcloud deploy**. The **raw_template** table retains all past templates and grows in size. To remove unused templates in the **raw_templates** table, create a daily cronjob that clears unused templates that exist in the database for longer than a day:

```
0 04 * * * /bin/heat-manage purge_deleted -g days 1
```

- The **openstack-heat-engine** and **openstack-heat-api** services might consume too many resources at times. If so, set **max_resources_per_stack=-1** in **/etc/heat/heat.conf** and restart the heat services:

  ```
  $ sudo systemctl restart openstack-heat-engine openstack-heat-api
  ```

- Sometimes the director might not have enough resources to perform concurrent node provisioning. The default is 10 nodes at the same time. To reduce the number of concurrent nodes, set the **max_concurrent_builds** parameter in **/etc/nova/nova.conf** to a value less than 10 and restart the nova services:

  ```
  $ sudo systemctl restart openstack-nova-api openstack-nova-scheduler
  ```

- Edit the **/etc/my.cnf.d/server.cnf** file. Some recommended values to tune include:

  max_connections

  Number of simultaneous connections to the database. The recommended value is 4096.

  innodb_additional_mem_pool_size

  The size in bytes of a memory pool the database uses to store data dictionary information and other internal data structures. The default is usually 8M and an ideal value is 20M for the undercloud.

  innodb_buffer_pool_size

  The size in bytes of the buffer pool, the memory area where the database caches table and index data. The default is usually 128M and an ideal value is 1000M for the undercloud.

  innodb_flush_log_at_trx_commit

  Controls the balance between strict ACID compliance for commit operations, and higher performance that is possible when commit-related I/O operations are rearranged and done in batches. Set to 1.

  innodb_lock_wait_timeout

  The length of time in seconds a database transaction waits for a row lock before giving up. Set to 50.

  innodb_max_purge_lag

  This variable controls how to delay INSERT, UPDATE, and DELETE operations when purge operations are lagging. Set to 10000.

  innodb_thread_concurrency

  The limit of concurrent operating system threads. Ideally, provide at least two threads for each CPU and disk resource. For example, if using a quad-core CPU and a single disk, use 10 threads.

- Ensure that heat has enough workers to perform an overcloud creation. Usually, this depends on how many CPUs the undercloud has. To manually set the number of workers, edit the **/etc/heat/heat.conf** file, set the **num_engine_workers** parameter to the number of workers you need (ideally 4), and restart the heat engine:

  ```
  $ sudo systemctl restart openstack-heat-engine
  ```

11.9. IMPORTANT LOGS FOR UNDERCLOUD AND OVERCLOUD

Use the following logs to find out information about the undercloud and overcloud when troubleshooting.

Table 11.2. Important Logs for the Undercloud

Information	Log Location
OpenStack Compute log	**/var/log/nova/nova-compute.log**
OpenStack Compute API interactions	**/var/log/nova/nova-api.log**
OpenStack Compute Conductor log	**/var/log/nova/nova-conductor.log**
OpenStack Orchestration log	**heat-engine.log**
OpenStack Orchestration API interactions	**heat-api.log**
OpenStack Orchestration CloudFormations log	**/var/log/heat/heat-api-cfn.log**
OpenStack Bare Metal Conductor log	**ironic-conductor.log**
OpenStack Bare Metal API interactions	**ironic-api.log**
Introspection	**/var/log/ironic-inspector/ironic-inspector.log**
OpenStack Workflow Engine log	**/var/log/mistral/engine.log**
OpenStack Workflow Executor log	**/var/log/mistral/executor.log**
OpenStack Workflow API interactions	**/var/log/mistral/api.log**

Table 11.3. Important Logs for the Overcloud

Information	Log Location
Cloud-Init Log	**/var/log/cloud-init.log**
Overcloud Configuration (Summary of Last Puppet Run)	**/var/lib/puppet/state/last_run_summary.yaml**
Overcloud Configuration (Report from Last Puppet Run)	**/var/lib/puppet/state/last_run_report.yaml**
Overcloud Configuration (All Puppet Reports)	**/var/lib/puppet/reports/overcloud-*/***
Overcloud Configuration (stdout from each Puppet Run)	**/var/run/heat-config/deployed/*-stdout.log**

Information	Log Location
Overcloud Configuration (stderr from each Puppet Run)	**/var/run/heat-config/deployed/*-stderr.log**
High availability log	**/var/log/pacemaker.log**

APPENDIX A. SSL/TLS CERTIFICATE CONFIGURATION

You can configure the undercloud to use SSL/TLS for communication over public endpoints. However, if using a SSL certificate with your own certificate authority, the certificate requires the configuration steps in the following section.

NOTE

For overcloud SSL/TLS certificate creation, see "Enabling SSL/TLS on the Overcloud" in the *Advanced Overcloud Customization* guide.

A.1. INITIALIZING THE SIGNING HOST

The signing host is the host that generates new certificates and signs them with a certificate authority. If you have never created SSL certificates on the chosen signing host, you might need to initialize the host so that it can sign new certificates.

The **/etc/pki/CA/index.txt** file stores records of all signed certificates. Check if this file exists. If it does not exist, create an empty file:

```
$ sudo touch /etc/pki/CA/index.txt
```

The **/etc/pki/CA/serial** file identifies the next serial number to use for the next certificate to sign. Check if this file exists. If it does not exist, create a new file with a new starting value:

```
$ echo '1000' | sudo tee /etc/pki/CA/serial
```

A.2. CREATING A CERTIFICATE AUTHORITY

Normally you sign your SSL/TLS certificates with an external certificate authority. In some situations, you might aim to use your own certificate authority. For example, you might aim to have an internal-only certificate authority.

For example, generate a key and certificate pair to act as the certificate authority:

```
$ openssl genrsa -out ca.key.pem 4096
$ openssl req  -key ca.key.pem -new -x509 -days 7300 -extensions v3_ca -out ca.crt.pem
```

The **openssl req** command asks for certain details about your authority. Enter these details.

This creates a certificate authority file called **ca.crt.pem**.

A.3. ADDING THE CERTIFICATE AUTHORITY TO CLIENTS

For any external clients aiming to communicate using SSL/TLS, copy the certificate authority file to each client that requires access your Red Hat OpenStack Platform environment. Once copied to the client, run the following command on the client to add it to the certificate authority trust bundle:

```
$ sudo cp ca.crt.pem /etc/pki/ca-trust/source/anchors/
$ sudo update-ca-trust extract
```

A.4. CREATING AN SSL/TLS KEY

Run the following commands to generate the SSL/TLS key (**server.key.pem**), which we use at different points to generate our undercloud or overcloud certificates:

```
$ openssl genrsa -out server.key.pem 2048
```

A.5. CREATING AN SSL/TLS CERTIFICATE SIGNING REQUEST

This next procedure creates a certificate signing request for either the undercloud or overcloud.

Copy the default OpenSSL configuration file for customization.

```
$ cp /etc/pki/tls/openssl.cnf .
```

Edit the custom **openssl.cnf** file and set SSL parameters to use for the director. An example of the types of parameters to modify include:

```
[req]
distinguished_name = req_distinguished_name
req_extensions = v3_req

[req_distinguished_name]
countryName = Country Name (2 letter code)
countryName_default = AU
stateOrProvinceName = State or Province Name (full name)
stateOrProvinceName_default = Queensland
localityName = Locality Name (eg, city)
localityName_default = Brisbane
organizationalUnitName = Organizational Unit Name (eg, section)
organizationalUnitName_default = Red Hat
commonName = Common Name
commonName_default = 192.168.0.1
commonName_max = 64

[ v3_req ]
# Extensions to add to a certificate request
basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
subjectAltName = @alt_names

[alt_names]
IP.1 = 192.168.0.1
DNS.1 = instack.localdomain
DNS.2 = vip.localdomain
DNS.3 = 192.168.0.1
```

Set the **commonName_default** to one of the following:

- If using an IP address to access over SSL/TLS, use the **undercloud_public_vip** parameter in **undercloud.conf**.
- If using a fully qualified domain name to access over SSL/TLS, use the domain name instead.

Edit the **alt_names** section to include the following entries:

- **IP** - A list of IP addresses for clients to access the director over SSL.
- **DNS** - A list of domain names for clients to access the director over SSL. Also include the Public API IP address as a DNS entry at the end of the **alt_names** section.

For more information about **openssl.cnf**, run **man openssl.cnf**.

Run the following command to generate certificate signing request (**server.csr.pem**):

```
$ openssl req -config openssl.cnf -key server.key.pem -new -out server.csr.pem
```

Make sure to include the SSL/TLS key you created in Section A.4, "Creating an SSL/TLS Key" for the **-key** option.

Use the **server.csr.pem** file to create the SSL/TLS certificate in the next section.

A.6. CREATING THE SSL/TLS CERTIFICATE

The following command creates a certificate for your undercloud or overcloud:

```
$ sudo openssl ca -config openssl.cnf -extensions v3_req -days 3650 -in server.csr.pem -out
server.crt.pem -cert ca.crt.pem -keyfile ca.key.pem
```

This command uses:

- The configuration file specifying the v3 extensions. Include this as the **-config** option.
- The certificate signing request from Section A.5, "Creating an SSL/TLS Certificate Signing Request" to generate the certificate and sign it through a certificate authority. Include this as the **-in** option.
- The certificate authority you created in Section A.2, "Creating a Certificate Authority", which signs the certificate. Include this as the **-cert** option.
- The certificate authority private key you created in Section A.2, "Creating a Certificate Authority". Include this as the **-keyfile** option.

This results in a certificate named **server.crt.pem**. Use this certificate in conjunction with the SSL/TLS key from Section A.4, "Creating an SSL/TLS Key" to enable SSL/TLS.

A.7. USING THE CERTIFICATE WITH THE UNDERCLOUD

Run the following command to combine the certificate and key together:

```
$ cat server.crt.pem server.key.pem > undercloud.pem
```

This creates a **undercloud.pem** file. You specify the location of this file for the **undercloud_service_certificate** option in your **undercloud.conf** file. This file also requires a special SELinux context so that the HAProxy tool can read it. Use the following example as a guide:

```
$ sudo mkdir /etc/pki/instack-certs
$ sudo cp ~/undercloud.pem /etc/pki/instack-certs/.
$ sudo semanage fcontext -a -t etc_t "/etc/pki/instack-certs(/.*)?"
$ sudo restorecon -R /etc/pki/instack-certs
```

Add the **undercloud.pem** file location to the **undercloud_service_certificate** option in the **undercloud.conf** file. For example:

```
undercloud_service_certificate = /etc/pki/instack-certs/undercloud.pem
```

In addition, make sure to add your certificate authority from Section A.2, “Creating a Certificate Authority” to the undercloud’s list of trusted Certificate Authorities so that different services within the undercloud have access to the certificate authority:

```
$ sudo cp ca.crt.pem /etc/pki/ca-trust/source/anchors/
$ sudo update-ca-trust extract
```

Continue installing the undercloud as per the instructions in Section 4.6, “Configuring the Director”.

APPENDIX B. POWER MANAGEMENT DRIVERS

Although IPMI is the main method the director uses for power management control, the director also supports other power management types. This appendix provides a list of the supported power management features. Use these power management settings for Section 5.1, “Registering Nodes for the Overcloud”.

B.1. 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 **pxe_drac**.

pm_user; pm_password

The DRAC username and password.

pm_addr

The IP address of the DRAC host.

B.2. 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 **pxe_ilo**.

pm_user; pm_password

The iLO username and password.

pm_addr

The IP address of the iLO interface.

- Edit the **/etc/ironic/ironic.conf** file and add **pxe_ilo** to the **enabled_drivers** option to enable this driver.
- The director also requires an additional set of utilities for iLo. Install the **python-proliantutils** package and restart the **openstack-ironic-conductor** service:

```
$ sudo yum install python-proliantutils
$ sudo systemctl restart openstack-ironic-conductor.service
```

- HP nodes must a 2015 firmware version for successful introspection. The director has been successfully tested with nodes using firmware version 1.85 (May 13 2015).
- Using a shared iLO port is not supported.

B.3. CISCO UNIFIED COMPUTING SYSTEM (UCS)

UCS from Cisco is a data center platform that unites compute, network, storage access, and virtualization resources. This driver focuses on the power management for bare metal systems connected to the UCS.

pm_type

Set this option to **pxe_ucs**.

pm_user; pm_password

The UCS username and password.

pm_addr

The IP address of the UCS interface.

pm_service_profile

The UCS service profile to use. Usually takes the format of **org-root/ls-[service_profile_name]**. For example:

```
"pm_service_profile": "org-root/ls-Nova-1"
```

- Edit the **/etc/ironic/ironic.conf** file and add **pxe_ucs** to the **enabled_drivers** option to enable this driver.

- The director also requires an additional set of utilities for UCS. Install the **python-UcsSdk** package and restart the **openstack-ironic-conductor** service:

```
$ sudo yum install python-UcsSdk
$ sudo systemctl restart openstack-ironic-conductor.service
```

B.4. FUJITSU INTEGRATED REMOTE MANAGEMENT CONTROLLER (IRMC)

Fujitsu's 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 **pxe_irmc**.

pm_user; pm_password

The username and password for the iRMC interface.

pm_addr

The IP address of the iRMC interface.

pm_port (Optional)

The port to use for iRMC operations. The default is 443.

pm_auth_method (Optional)

The authentication method for iRMC operations. Use either **basic** or **digest**. The default is **basic**

pm_client_timeout (Optional)

Timeout (in seconds) for iRMC operations. The default is 60 seconds.

pm_sensor_method (Optional)

Sensor data retrieval method. Use either **ipmitool** or **scci**. The default is **ipmitool**.

- Edit the **/etc/ironic/ironic.conf** file and add **pxe_irmc** to the **enabled_drivers** option to enable this driver.

- The director also requires an additional set of utilities if you enabled SCCI as the sensor method. Install the **python-scciclient** package and restart the **openstack-ironic-conductor** service:

```
$ yum install python-scciclient
$ sudo systemctl restart openstack-ironic-conductor.service
```

B.5. SSH AND VIRSH

The director can access a host running libvirt through SSH and use virtual machines as nodes. The director uses virsh to control the power management of these nodes.

> **IMPORTANT**
>
> This option is available for testing and evaluation purposes only. It is not recommended for Red Hat OpenStack Platform enterprise environments.

pm_type

Set this option to **pxe_ssh**.

pm_user; pm_password

The SSH username and contents of the SSH private key. If using the CLI tools to register your nodes, the private key must be on one line with new lines replaced with escape characters (**\n**). For example:

```
-----BEGIN RSA PRIVATE KEY-----\nMIIEogIBAAKCAQEA .... kk+WXt9Y=\n-----END RSA
PRIVATE KEY-----
```

Add the SSH public key to the libvirt server's **authorized_keys** collection.

pm_addr

The IP address of the virsh host.

- The server hosting libvirt requires an SSH key pair with the public key set as the **pm_password** attribute.

- Ensure the chosen **pm_user** has full access to the libvirt environment.

B.6. FAKE PXE DRIVER

This driver provides a method to use bare metal devices without power management. This means the director does not control the registered bare metal devices and as such require manual control of power at certain points in the introspect and deployment processes.

> **IMPORTANT**
>
> This option is available for testing and evaluation purposes only. It is not recommended for Red Hat OpenStack Platform enterprise environments.

pm_type

Set this option to **fake_pxe**.

- This driver does not use any authentication details because it does not control power management.
- Edit the **/etc/ironic/ironic.conf** file and add **fake_pxe** to the **enabled_drivers** option to enable this driver. Restart the baremetal services after editing the file:

  ```
  $ sudo systemctl restart openstack-ironic-api openstack-ironic-conductor
  ```

- When performing introspection on nodes, manually power the nodes after running the **openstack baremetal introspection bulk start** command.
- When performing overcloud deployment, check the node status with the **ironic node-list** command. Wait until the node status changes from **deploying** to **deploy wait-callback** and then manually power the nodes.
- After the overcloud provisioning process completes, reboot the nodes. To check the completion of provisioning, check the node status with the **ironic node-list** command, wait until the node status changes to **active**, then manually reboot all overcloud nodes.

APPENDIX C. WHOLE DISK IMAGES

The main overcloud image is a flat partition image. This means it contains no partitioning information or bootloader on the images itself. The director uses a seperate kernel and ramdisk when booting and creates a basic partitioning layout when writing the overcloud image to disk. However, you can create a whole disk image, which includes a partitioning layout and bootloader.

C.1. CREATING WHOLE DISK IMAGES

Creating a whole disk image from the **overcloud-full.qcow2** flat partition image involves the following steps:

1. Open the **overcloud-full** flat partition as a base for our whole disk image.
2. Create a new whole disk image with the desired size. This example uses a 10 GB image.
3. Create partitions and volumes on the whole disk image. Create as many partitions and volumes necessary for your desired whole disk image. This example creates an isolated partition for **boot** and logical volumes for the other content in the filesystem.
4. Create the initial filesystems on the partitions and volumes.
5. Mount flat partition filesystem and copy content to the right partitions on the whole disk image.
6. Generate the **fstab** content and save it to **/etc/fstab** on the whole disk image.
7. Unmount all the filesystems.
8. Mount the partitions on the whole disk image only. Start with the root partition mounted at / and mount the other partition in their respective directories.
9. Install the bootloader using shell commands to execute **grub2-install** and **grub2-mkconfig** on the whole disk image. This installs the **grub2** bootloader in the whole disk image.
10. Update **dracut** to provide support for logical volume management
11. Unmount all the filesystems and close the image

Manually Creating a Whole Disk Image

A recommended tool for creating images is **guestfish**, which you install using the following command:

```
$ sudo yum install -y guestfish
```

Once installed, run the **guestfish** interactive shell:

```
$ guestfish

Welcome to guestfish, the guest filesystem shell for
editing virtual machine filesystems and disk images.

Type: 'help' for help on commands
      'man' to read the manual
      'quit' to quit the shell

><fs>
```

For more information on using **guestfish** see "The Guestfish Shell" in the *Virtualization Deployment and Administration Guide* for Red Hat Enterprise Linux 7.

C.2. AUTOMATICALLY CREATING A WHOLE DISK IMAGE

The following Python script uses the **guestfish** library to automatically generate the whole disk image.

```
#!/usr/bin/env python
import guestfs
import os

# remove old generated drive
try:
  os.unlink("/home/stack/images/overcloud-full-partitioned.qcow2")
except:
  pass

g = guestfs.GuestFS(python_return_dict=True)

# import old and new images
print("Creating new repartitioned image")
g.add_drive_opts("/home/stack/images/overcloud-full.qcow2", format="qcow2", readonly=1)
g.disk_create("/home/stack/images/overcloud-full-partitioned.qcow2", "qcow2", 10.2 * 1024 * 1024 *
1024) #10.2G
g.add_drive_opts("/home/stack/images/overcloud-full-partitioned.qcow2", format="qcow2",
readonly=0)
g.launch()

# create the partitions for new image
print("Creating the initial partitions")
g.part_init("/dev/sdb", "mbr")
g.part_add("/dev/sdb", "primary", 2048, 616448)
g.part_add("/dev/sdb", "primary", 616449, -1)

g.pvcreate("/dev/sdb2")
g.vgcreate("vg", ['/dev/sdb2', ])
g.lvcreate("var", "vg", 5 * 1024)
g.lvcreate("tmp", "vg", 500)
g.lvcreate("swap", "vg", 250)
g.lvcreate("home", "vg", 100)
g.lvcreate("root", "vg", 4 * 1024)
g.part_set_bootable("/dev/sdb", 1, True)

# add filesystems to volumes
print("Adding filesystems")
ids = {}
keys = [ 'var', 'tmp', 'swap', 'home', 'root' ]
volumes = ['/dev/vg/var', '/dev/vg/tmp', '/dev/vg/swap', '/dev/vg/home', '/dev/vg/root']
swap_volume = volumes[2]

count = 0
for volume in volumes:
  if count!=2:
    g.mkfs('ext4', volume)
    ids[keys[count]] = g.vfs_uuid(volume)
  count +=1

# create filesystem on boot and swap
g.mkfs('ext4', '/dev/sdb1')
g.mkswap_opts(volumes[2])
ids['swap'] = g.vfs_uuid(volumes[2])

# mount drives and copy content
print("Start copying content")
g.mkmountpoint('/old')
g.mkmountpoint('/root')
g.mkmountpoint('/boot')
g.mkmountpoint('/home')
g.mkmountpoint('/var')
g.mount('/dev/sda', '/old')

g.mount('/dev/sdb1', '/boot')
g.mount(volumes[4], '/root')
g.mount(volumes[3], '/home')
g.mount(volumes[0], '/var')

# copy content to root
results = g.ls('/old/')
for result in results:
  if result not in ('boot', 'home', 'tmp', 'var'):
    print("Copying %s to root" % result)
    g.cp_a('/old/%s' % result, '/root/')

# copy extra content
folders_to_copy = ['boot', 'home', 'var']
for folder in folders_to_copy:
  results = g.ls('/old/%s/' % folder)
  for result in results:
    print("Copying %s to %s" % (result, folder))
    g.cp_a('/old/%s/%s' % (folder, result),
           '/%s/' % folder)

# create /etc/fstab file
print("Generating fstab content")
fstab_content = """
UUID={boot_id} /boot ext4 defaults 0 2
UUID={root_id} / ext4 defaults 0 1
UUID={swap_id} none swap sw 0 0
UUID={tmp_id} /tmp ext4 defaults 0 2
UUID={home_id} /home ext4 defaults 0 2
UUID={var_id} /var ext4 defaults 0 2
""".format(
  boot_id=g.vfs_uuid('/dev/sdb1'),
  root_id=ids['root'],
  swap_id=ids['swap'],
  tmp_id=ids['tmp'],
  home_id=ids['home'],
  var_id=ids['var'])

g.write('/root/etc/fstab', fstab_content)
```

```
# unmount filesystems
g.umount('/root')
g.umount('/boot')
g.umount('/old')
g.umount('/var')

# mount in the right directories to install bootloader
print("Installing bootloader")
g.mount(volumes[4], '/')
g.mkdir('/boot')
g.mkdir('/var')
g.mount('/dev/sdb1', '/boot')
g.mount(volumes[0], '/var')

# do a selinux relabel
g.selinux_relabel('/etc/selinux/targeted/contexts/files/file_contexts', '/', force=True)
g.selinux_relabel('/etc/selinux/targeted/contexts/files/file_contexts', '/var', force=True)

g.sh('grub2-install --target=i386-pc /dev/sdb')
g.sh('grub2-mkconfig -o /boot/grub2/grub.cfg')

# create dracut.conf file
dracut_content = """
add_dracutmodules+="lvm crypt"
"""
g.write('/etc/dracut.conf', dracut_content)

# update initramfs to include lvm and crypt
kernels = g.ls('/lib/modules')
for kernel in kernels:
  print("Updating dracut to include modules in kernel %s" % kernel)
  g.sh('dracut -f /boot/initramfs-%s.img %s --force' % (kernel, kernel))
g.umount('/boot')
g.umount('/var')
g.umount('/')

# close images
print("Finishing image")
g.shutdown()
g.close()
```

Save this script as a executable file on the undercloud and run it as the **stack** user:

```
$ ./whole-disk-image.py
```

This automatically creates the whole disk image from the flat partition image. Once the whole disk image creation completes, replace the old **overcloud-full.qcow2** image:

```
$ mv ~/images/overcloud-full.qcow2 ~/images/overcloud-full-old.qcow2
$ cp ~/images/overcloud-full-partitioned.qcow2 ~/images/overcloud-full.qcow2
```

You can now upload the whole disk image along with your other images.

C.3. ENCRYPTING VOLUMES ON WHOLE DISK IMAGES

You can also use **guestfish** to encrypt volumes on your whole disk image. This requires using the **luks-format** subcommand, which erases the current volume and creates an encrypted volume.

The following Python script opens the **overcloud-full-partitioned.qcow2** image created previously, removes the current **home** volume (which is empty), and replaces it with an encrypted **home** volume:

```
#!/usr/bin/env python
import binascii
import guestfs

g = guestfs.GuestFS(python_return_dict=True)
g.add_drive_opts("/home/stack/images/overcloud-full-partitioned.qcow2", format="qcow2",
readonly=0)
g.launch()

random_content = binascii.b2a_hex(os.urandom(1024))
g.luks_format('/dev/vg/home', random_content, 0)
g.luks_open('/dev/vg/home', random_content, 'cryptedhome')
g.vgscan()
g.vg_activate_all(True)
g.mkfs('ext4', '/dev/mapper/cryptedhome')
g.mount('/dev/vg/root','/')

volumes = lvs()
volumes.remove('/dev/vg/home')
volumes.remove('/dev/vg/root')
volumes.remove('/dev/vg/swap')
fstab_content = []
fstab_content.append('UUID=%s /boot ext4 defaults 0 2' % g.vfs_uuid('/dev/sda1'))
fstab_content.append('UUID=%s / ext4 defaults 0 1' % g.vfs_uuid('/dev/vg/root'))
fstab_content.append('UUID=%s none swap sw 0 0' % g.vfs_uuid('/dev/vg/swap'))
fstab_content.append('/dev/mapper/cryptedhome /home ext4 defaults 0 1')
for volume in volumes:
  volume_name = volume.replace('/dev/vg/', '')
  fstab_content.append('UUID=%s /%s ext4 defaults 0 2' % (g.vfs_uuid(volume), volume_name))

g.write('/etc/fstab', '\n'.join(fstab_content))
print '\n'.join(fstab_content)

g.write('/root/home_keyfile', random_content)
g.chmod(0400, '/root/home_keyfile')

mapper = """
home UUID={home_id} /root/home_keyfile
""".format(home_id=g.vfs_uuid('/dev/mapper/cryptedhome'))
g.write('/etc/crypttab', mapper)

g.luks_close('/dev/mapper/cryptedhome')
g.selinux_relabel('/etc/selinux/targeted/contexts/files/file_contexts', '/', force=True)
g.shutdown()
g.close()
```

This script also:

- Creates a key (**random_content**)
- Regenerates the **/etc/fstab** file with the new encrypted volume
- Saves the encryption key at **/root/home_keyfile**
- Generates a **crypttab** file to automatically decrypt the volume using the **/root/home_keyfile**)

Use this script as an example to create encrypted volumes as part of your whole disk image creation process.

C.4. UPLOADING WHOLE DISK IMAGES

To upload a whole disk image, use the **--whole-disk-image** option with the image upload command. For example:

```
$ openstack overcloud image upload --whole-disk --image-path /home/stack/images
```

This command uploads the images from **/home/stack/images** but treats the **overcloud-full.qcow2** file as a whole disk image. This means you must rename the desired whole disk image to **overcloud-full.qcow2** before running the image upload command.

APPENDIX D. ALTERNATIVE BOOT MODES

The default boot mode for nodes is BIOS over iPXE. The following sections outline some alternative boot modes for the director to use when provisioning and inspecting nodes.

D.1. STANDARD PXE

The iPXE boot process uses HTTP to boot the introspection and deployment images. Older systems might only support a standard PXE boot, which boots over TFTP.

To change from iPXE to PXE, edit the **undercloud.conf** file on the director host and set **ipxe_enabled** to **False**:

```
ipxe_enabled = False
```

Save this file and run the undercloud installation:

```
$ openstack undercloud install
```

For more information on this process, see the article "Changing from iPXE to PXE in Red Hat OpenStack Platform director".

D.2. UEFI BOOT MODE

The default boot mode is the legacy BIOS mode. Newer systems might require UEFI boot mode instead of the legacy BIOS mode. In this situation, set the following in your **undercloud.conf** file:

```
ipxe_enabled = True
inspection_enable_uefi = True
```

Save this file and run the undercloud installation:

```
$ openstack undercloud install
```

Set the boot mode to **uefi** for each registered node. For example, to add or replace the existing **boot_mode** parameters in the **capabilities** property:

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

NOTE

Check that you have retained the **profile** and **boot_option** capabilities with this command.

In addition, set the boot mode to **uefi** for each flavor. For example:

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

APPENDIX E. AUTOMATIC PROFILE TAGGING

The introspection process performs a series of benchmark tests. The director saves the data from these tests. You can create a set of policies that use this data in various ways. For example:

- The policies can identify and isolate underperforming or unstable nodes from use in the overcloud.
- The policies can define whether to automatically tag nodes into specific profiles.

E.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*.

Description

This 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. For field types, see Section E.4, “Automatic Profile Tagging Properties”

op

Defines the operation to use for the evaluation. This includes the following:

- **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 includes:

- **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. If not, the condition returns false.

Example:

```
"conditions": [
  {
    "field": "local_gb",
    "op": "ge",
    "value": 1024
  }
],
```

Actions

An action is performed if the condition returns as true. It 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 (e.g. **/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 accordingly. The existing value for this same capability is replaced. 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"
  }
]
```

E.2. POLICY FILE EXAMPLE

The following is an example JSON file (**rules.json**) with the introspection rules to apply:

–

```
[
  {
    "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 is 4096 MiB. Such rules can be applied to exclude nodes that should not become part of your cloud.
- Nodes with hard drive size 1 TiB and bigger are assigned the swift-storage profile unconditionally.
- Nodes with hard drive less than 1 TiB but more than 40 GiB can be either Compute or Controller nodes. We assign two capabilities (**compute_profile** and **control_profile**) so that the **openstack overcloud profiles match** command can later make the final choice. For that to work, we remove the existing profile capability, otherwise it will have priority.

Other nodes are not changed.

NOTE

Using introspection rules to assign the **profile** capability always overrides the existing value. However, **[PROFILE]_profile** capabilities are ignored for nodes with an existing profile capability.

E.3. IMPORTING POLICY FILES

Import the policy file into the director with the following command:

```
$ openstack baremetal introspection rule import rules.json
```

Then run the introspection process.

```
$ openstack baremetal introspection bulk start
```

After introspection completes, check the nodes and their assigned profiles:

```
$ openstack overcloud profiles list
```

If you made a mistake in introspection rules, you can delete them all:

```
$ openstack baremetal introspection rule purge
```

E.4. AUTOMATIC PROFILE TAGGING PROPERTIES

Automatic Profile Tagging evaluates the following node properties for the **field** attribute for each condition:

Property	Description
memory_mb	The amount of memory for the node in MB.
cpus	The total number of cores for the node's CPUs.
cpu_arch	The architecture of the node's CPUs.
local_gb	The total storage space of the node's root disk. See Section 5.4, "Defining the Root Disk for Nodes" for more information about setting the root disk for a node.

APPENDIX F. FIREFOX SERVER EXCEPTIONS FOR UI ACCESS

Accessing the director’s web UI through Firefox requires certain server identity exceptions to allow OpenStack Platform services. This includes the following service URLs:

URL	service
https://<undercloud>:13000	OpenStack Identity (keystone)
https://<undercloud>:13004	OpenStack Orchestration (heat)
https://<undercloud>:13385	OpenStack Bare Metal (ironic)
https://<undercloud>:13989	OpenStack Workflow Service (mistral)
https://<undercloud>:13808	OpenStack Object Storage (swift)
https://<undercloud>:9000	OpenStack Messaging Service (zaqar)

Replace **<undercloud>** with the Public API IP address of your undercloud.

To add an server identity exception for a URL in Firefox:

1. Navigate to the **Preferences**.
2. Navigate to **Certificates** > **View Certificates**.
3. Click on the **Servers** tab and click **Add Exception**.
4. A window appears asking for the URL of the server. Enter a service URL and port from the list above and click **Get Certificate**.
5. Click **Confirm Security Exception**.

Continue adding exceptions for all service URLs.

APPENDIX G. SECURITY ENHANCEMENTS

The following sections provide some suggestions to harden the security of your undercloud.

G.1. CHANGING THE SSL/TLS CIPHER AND RULES FOR HAPROXY

If you enabled SSL/TLS in the undercloud (see Section 4.6, "Configuring the Director"), you might want to harden the SSL/TLS ciphers and rules used with the HAProxy configuration. This helps 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 aim 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 running **openstack undercloud install**:

```
[DEFAULT]
...
hieradata_override = haproxy-hiera-overrides.yaml
...
```