Red Hat OpenStack Platform 9

Director Installation and Usage

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

Last Updated: 2018-08-10

Red Hat OpenStack Platform 9 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

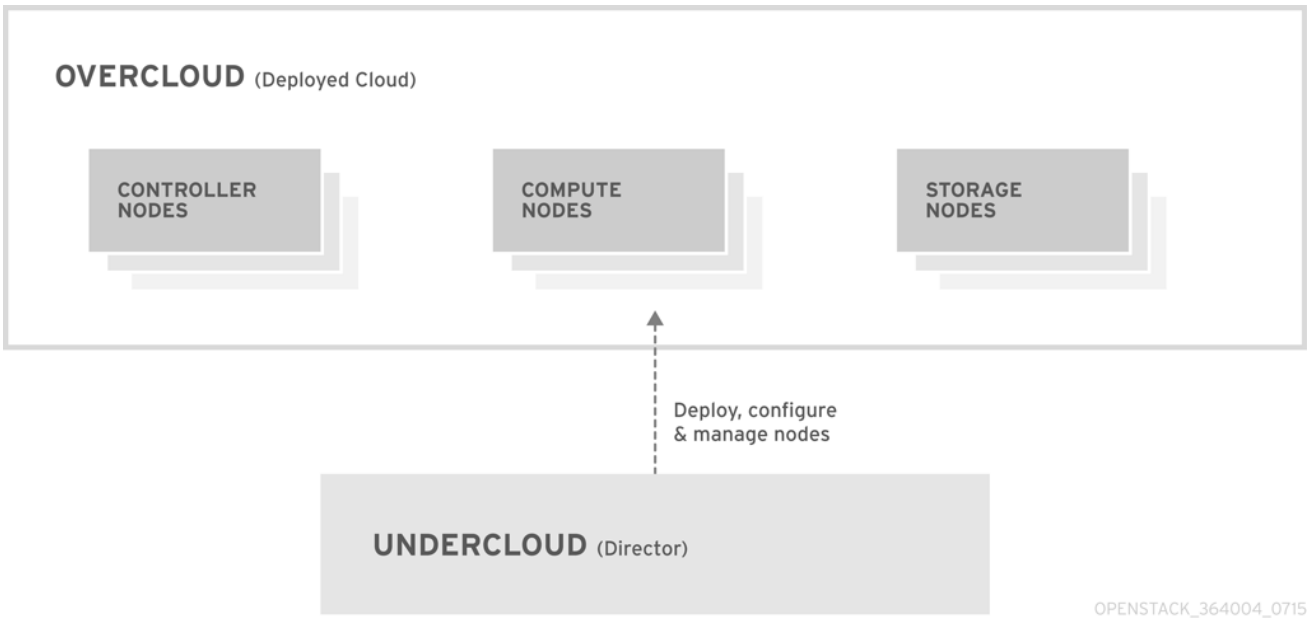
This guide explains how to install Red Hat OpenStack Platform 9 in an enterprise environment using the Red Hat OpenStack Platform Director. This includes installing the director, planning your environment, and creating an OpenStack environment with the director.

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 following functions:

- Environment planning - The Undercloud provides planning functions for users to assign Red Hat OpenStack Platform roles, including Compute, Controller, and various storage roles.
- Bare metal system control - The Undercloud uses the 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.
- Orchestration - The Undercloud provides and reads a set of YAML templates to create an OpenStack environment.

The Red Hat OpenStack Platform director performs these Undercloud functions through a terminal-based command line interface.

The Undercloud consists of the following 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 Identity (keystone) - Provides authentication and authorization for the director's components.
- OpenStack Objecdt 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 one or more of the following node types:

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)
- 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:

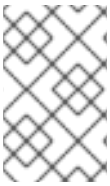
- Ceph Storage nodes - Used to form storage clusters. Each node contains a Ceph Object Storage Daemon (OSD). In addition, the director installs Ceph Monitor onto the Controller nodes in situations where it deploys Ceph Storage nodes.
- Block storage (cinder) - Used as external block storage for HA Controller 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 recommended best practices for installing 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 7.2 (or later) installed as the host operating system.

IMPORTANT

Ensure the Undercloud's file system only contains a root and swap partitions if using Logical Volume Management (LVM). For more information, see the Red Hat Customer Portal article "Director node fails to boot after undercloud installation".

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 remote connectivity to all nodes. 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. For more information on network isolation, see Section 6.2, "Isolating Networks".

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”.

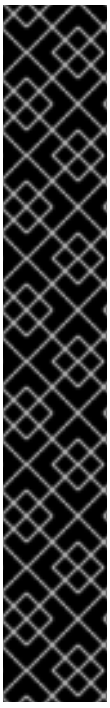
NOTE

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.
- 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:

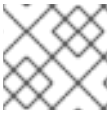
- Use network segmentation to mitigate network movement and isolate sensitive data; a flat network is much less secure.
- Restrict services access and ports to a minimum.
- Ensure proper firewall rules and password usage.
- Ensure that SELinux is enabled.

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 Red Hat 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 16 GB. However, the amount of recommended memory depends on the number of CPU cores. Use the following calculations as guidance:

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

A minimum of 40 GB of available disk space.

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 Red Hat OpenStack Platform environment.

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 a disk 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.
  This guide contains the necessary instructions to map your Ceph Storage disks into the director.

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.

IMPORTANT

The director does not create partitions on the journal disk. You must manually create these journal partitions before the Director can deploy the Ceph Storage nodes.

The Ceph Storage OSDs and journals partitions require GPT disk labels, which you also configure prior to customization. For example, use the following command on the potential Ceph Storage host to create a GPT disk label for a disk or partition:

```
# parted [device] mklabel gpt
```

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:

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 9 for RHEL 7 (RPMs)	**rhel-7-server-openstack-9-rpms**	Core Red Hat OpenStack Platform repository. Also contains packages for Red Hat OpenStack Platform director.
Red Hat Ceph Storage OSD 1.3 for Red Hat Enterprise Linux 7 Server (RPMs)	**rhel-7-server-rhceph-1.3-osd-rpms**	(For Ceph Storage Nodes) Repository for Ceph Storage Object Storage daemon. Installed on Ceph Storage nodes.
Red Hat Ceph Storage MON 1.3 for Red Hat Enterprise Linux 7 Server (RPMs)	**rhel-7-server-rhceph-1.3-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 1.3 for Red Hat Enterprise Linux 7 Workstation (RPMs)	**rhel-7-server-rhceph-1.3-tools-rpms**	Provides tools for nodes to communicate with the Ceph Storage cluster. This repository should be enabled for all 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

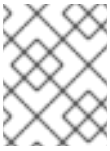
The following section provides some guidelines on planning various aspects of your Red Hat OpenStack Platform environment. This includes defining node roles, planning your network topology, and storage.

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 Overcloud configurations 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

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

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 backend connect over the Storage Management network, since they do not interact with Ceph directly but rather use the frontend service. Note that the RBD driver is an exception, as this traffic connects directly to Ceph.	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

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.

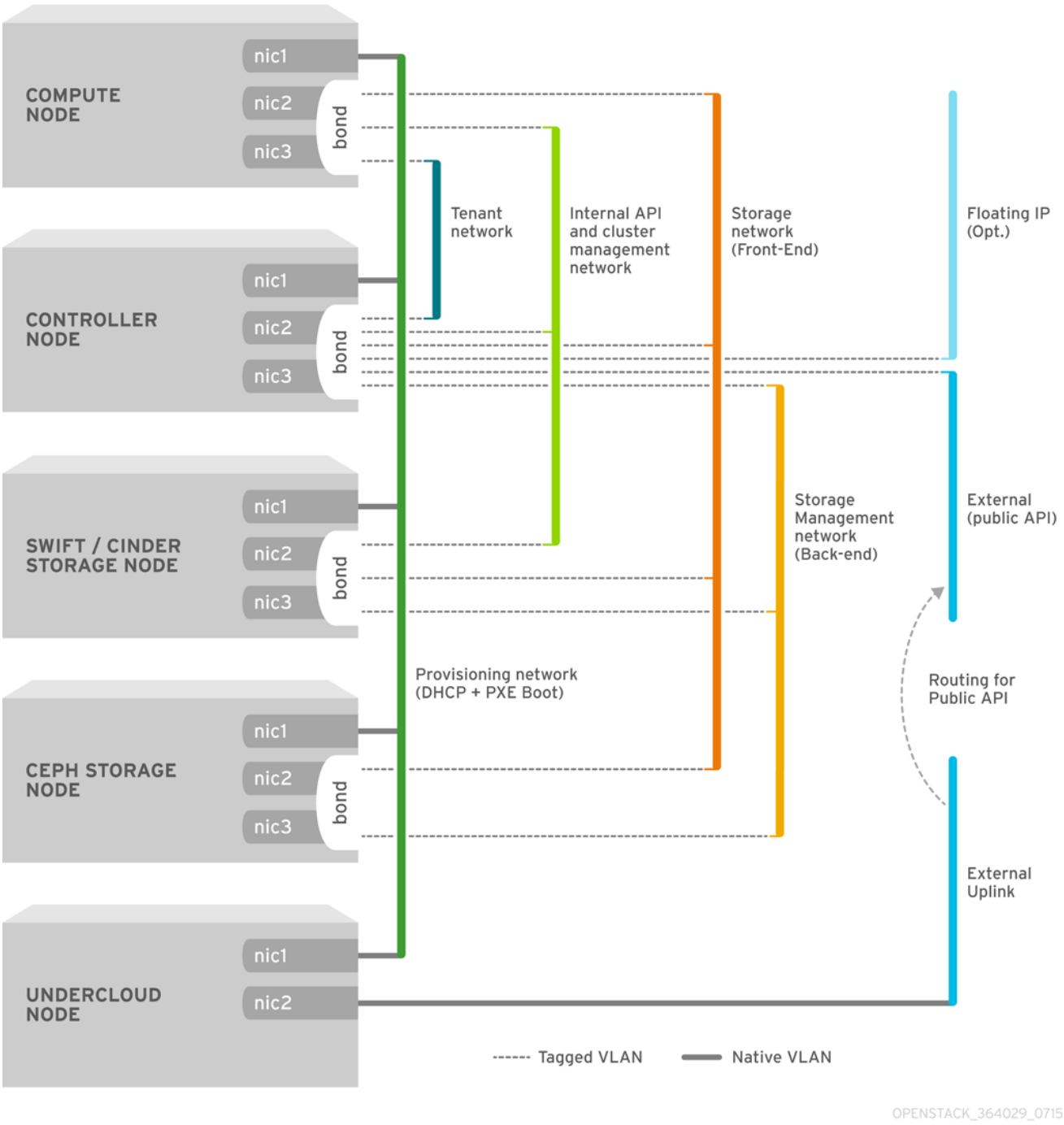
The director provides a method for mapping six of these traffic types to certain subnets or VLANs. These traffic types include:

- Internal API
- Storage

- Storage Management
- Tenant Networks
- External
- Management

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 - Storage Management Network 7 - External, Floating IP (mapped after Overcloud creation)	3 (includes 2 bonded interfaces)	7

3.3. PLANNING STORAGE

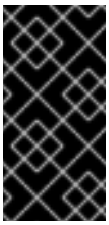
The director provides different storage options for the Overcloud environment. This includes:

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 backend 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 for the Red Hat OpenStack Platform director.

```
$ sudo subscription-manager list --available --all
```

Use the pool ID located in the previous step to attach the Red Hat OpenStack Platform 9 entitlements:

```
$ sudo subscription-manager attach --pool=pool_id
```

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-openstack-9-rpms --
enable=rhel-7-server-openstack-9-director-rpms --enable=rhel-7-server-rh-
common-rpms
```

These repositories contain packages the director installation requires.

IMPORTANT

Only enable the repositories listed above. Additional repositories can cause package and software conflicts. Do not enable any additional repositories.

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 basic template contains the following parameters:

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 defined for the director's Public API. Use an IP address on the Provisioning network that does not conflict with any other IP addresses or address ranges. For example, **192.0.2.2**. The director configuration attaches this IP address to its software bridge as a routed IP address, which uses the **/32** netmask.

undercloud_admin_vip

The IP address defined for the director's Admin API. Use an IP address on the Provisioning network that does not conflict with any other IP addresses or address ranges. For example, **192.0.2.3**. The director configuration attaches this IP address to its 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.

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.

network_cidr

The network that the director uses to manage Overcloud instances. This is the Provisioning network. 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.

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.

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**.

ipxe_deploy

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 "Changing from iPXE to PXE in Red Hat OpenStack Platform director" on the Red Hat Customer Portal.

store_events

Defines whether to store events in Ceilometer on the Undercloud.

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.

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-9.0.tar
/usr/share/rhosp-director-images/ironic-python-agent-latest-9.0.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
```

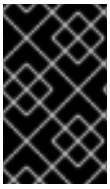
4.8. SETTING A NAMESERVER ON THE UNDERCLOUD'S NEUTRON SUBNET

Overcloud nodes require a nameserver so that they can resolve hostnames through DNS. For a standard Overcloud without network isolation, the nameserver is defined using the Undercloud's **neutron** subnet. Use the following commands to define the nameserver for the environment:

```
$ neutron subnet-list
$ neutron subnet-update [subnet-uuid] --dns-nameserver [nameserver-ip]
```

View the subnet to verify the nameserver:

```
$ neutron 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 templates. This is covered in the advanced configuration scenario in Section 6.2, “Isolating Networks”.

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

This chapter provides the basic configuration steps for an enterprise-level OpenStack Platform environment. 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 Chapter 6, *Configuring Advanced Customizations for the Overcloud*.

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. For more information, see Section 6.2, “Isolating Networks”.

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"
            ],
            "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"
            ],
            "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:

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

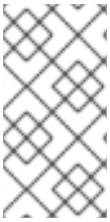
(Optional) The amount of memory in MB.

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, save the file to the **stack** user's home directory (**/home/stack/instackenv.json**), then import it into the director using the following command:

```
$ openstack baremetal import --json ~/instackenv.json
```

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

Assign the kernel and ramdisk images to all nodes:

```
$ openstack baremetal configure boot
```

The nodes are now registered and configured in the director. View a list of these nodes in the CLI:

```
$ ironic 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 C, *Automatic Profile Tagging*. Alternatively, you can manually tag nodes into profiles as per the instructions in Section 5.3, "Tagging Nodes into Profiles".

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

```
$ openstack baremetal introspection bulk start
```

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:

```
$ ironic node-set-provision-state [NODE UUID] manage
$ openstack baremetal introspection start [NODE UUID]
$ ironic node-set-provision-state [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. 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 C, *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:

```
$ ironic node-update 58c3d07e-24f2-48a7-bbb6-6843f0e8ee13 add
properties/capabilities='profile:compute,boot_option:local'
$ ironic node-update 1a4e30da-b6dc-499d-ba87-0bd8a3819bc0 add
properties/capabilities='profile:control,boot_option:local'
```

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 the boot mode for each node.

IMPORTANT

The director currently does not support UEFI boot mode.

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

```
$ openstack overcloud profiles list
```

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.
- **`wwn`** (String): Unique storage identifier.
- **`hctl`** (String): Host:Channel:Target:Lun for SCSI.
- **`size`** (Integer): Size of the device in GB.

In this example, you specify the drive to deploy the Overcloud image using the serial number of the disk to determine the root device.

First, collect a copy of each node's hardware information that the director obtained from the introspection. This information is stored in the OpenStack Object Storage server (swift). Download this information to a new directory:

```
$ mkdir swift-data
$ cd swift-data
$ export IRONIC_DISCOVERD_PASSWORD=`sudo grep admin_password /etc/ironic-
inspector/inspector.conf | awk '! /^#/ {print $NF}' | awk -F'=' '{print
$2}'`
$ for node in $(ironic node-list | awk '!/UUID/ {print $2}'); do swift -U
service:ironic -K $IRONIC_DISCOVERD_PASSWORD download ironic-inspector
inspector_data-$node; done
```

This downloads the data from each **`inspector_data`** object from introspection. All objects use the node UUID as part of the object name:

```
$ ls -1
inspector_data-15fc0edc-eb8d-4c7f-8dc0-a2a25d5e09e3
inspector_data-46b90a4d-769b-4b26-bb93-50eaefcdb3f4
inspector_data-662376ed-faa8-409c-b8ef-212f9754c9c7
inspector_data-6fc70fe4-92ea-457b-9713-eed499eda206
inspector_data-9238a73a-ec8b-4976-9409-3fcff9a8dca3
inspector_data-9cbfe693-8d55-47c2-a9d5-10e059a14e07
inspector_data-ad31b32d-e607-4495-815c-2b55ee04cdb1
inspector_data-d376f613-bc3e-4c4b-ad21-847c4ec850f8
```

Check the disk information for each node. The following command displays each node ID and the disk information:

```
$ for node in $(ironic node-list | awk '!/UUID/ {print $2}'); do echo
"NODE: $node" ; cat inspector_data-$node | jq '.inventory.disks' ; echo "-
----" ; done
```

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

```
NODE: 46b90a4d-769b-4b26-bb93-50eaefcdb3f4
[
  {
    "size": 1000215724032,
    "vendor": "ATA",
    "name": "/dev/sda",
    "model": "WDC WD1002F9YZ",
    "wwn": "0x0000000000000001",
    "serial": "WD-000000000001"
  },
  {
    "size": 1000215724032,
    "vendor": "ATA",
    "name": "/dev/sdb",
    "model": "WDC WD1002F9YZ",
    "wwn": "0x0000000000000002",
    "serial": "WD-000000000002"
  },
  {
    "size": 1000215724032,
    "vendor": "ATA",
    "name": "/dev/sdc",
    "model": "WDC WD1002F9YZ",
    "wwn": "0x0000000000000003",
    "serial": "WD-000000000003"
  },
]
```

For this example, set the root device to disk 2, which has **`WD-000000000002`** as the serial number. This requires a change to the **`root_device`** parameter for the node definition:

```
$ ironic node-update 97e3f7b3-5629-473e-a187-2193ebe0b5c7 add
properties/root_device='{"serial": "WD-000000000002"}'
```

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.

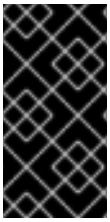
IMPORTANT

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

5.5. COMPLETING BASIC CONFIGURATION

This concludes the required steps for basic configuration of your Overcloud. You can now either:

- Customize your environment using advanced configuration step. See Chapter 6, *Configuring Advanced Customizations for the Overcloud* for more information.
- Or deploy a basic Overcloud. See Chapter 7, *Creating the Overcloud* for more information.

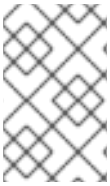


IMPORTANT

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. For example, see Section 6.7, “Configuring NFS Storage” for configuring an NFS share for block storage.

CHAPTER 6. CONFIGURING ADVANCED CUSTOMIZATIONS FOR THE OVERCLOUD

This chapter follows on from Chapter 5, *Configuring Basic Overcloud Requirements*. At this point, the director has registered the nodes and configured the necessary services for Overcloud creation. Now you can customize your Overcloud using the methods in this chapter.



NOTE

The examples in this chapter are optional steps for configuring the Overcloud. These steps are only required to provide the Overcloud with additional functionality. Use only the steps that apply to the needs of your environment.

6.1. UNDERSTANDING HEAT TEMPLATES

The custom configurations in this chapter use Heat templates and environment files to define certain aspects of the Overcloud, such as network isolation and network interface configuration. This section provides a basic introduction to heat templates so that you can understand the structure and format of these templates in the context of the Red Hat OpenStack Platform director.

6.1.1. Heat Templates

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

The structure of a Heat template has three main sections:

- **Parameters** - These are settings passed to heat, which provides a way to customize a stack, and any default values for parameters without passed values. These are defined in the **`parameters`** section of a template.
- **Resources** - These are the specific objects to create and configure as part of a stack. OpenStack contains a set of core resources that span across all components. These are defined in the **`resources`** section of a template.
- **Output** - These are values passed from heat after the stack's creation. You can access these values either through the heat API or client tools. These are defined in the **`output`** section of a template.

Here is an example of a basic heat template:

```
heat_template_version: 2013-05-23

description: > A very basic Heat template.

parameters:
  key_name:
    type: string
    default: lars
    description: Name of an existing key pair to use for the instance
  flavor:
```

```
    type: string
    description: Instance type for the instance to be created
    default: m1.small
  image:
    type: string
    default: cirros
    description: ID or name of the image to use for the instance

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

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

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

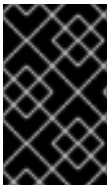
When Heat processes a template it creates a stack for the template and a set of child stacks for resource templates. This creates a hierarchy of stacks that descend from the main stack you define with your template. You can view the stack hierarchy using this following command:

```
$ heat stack-list --show-nested
```

6.1.2. Environment Files

An environment file is a special type of template that provides customization for your Heat templates. This includes three key parts:

- **Resource Registry** - This section defines custom resource names, linked to other heat templates. This essentially provides a method to create custom resources that do not exist within the core resource collection. These are defined in the **`resource_registry`** section of an environment file.
- **Parameters** - These are common settings you apply to the top-level template's parameters. For example, if you have a template that deploys nested stacks, such as resource registry mappings, the parameters only apply to the top-level template and not templates for the nested resources. Parameters are defined in the **`parameters`** section of an environment file.
- **Parameter Defaults** - These parameters modify the default values for parameters in all templates. For example, if you have a Heat template that deploys nested stacks, such as resource registry mappings,the parameter defaults apply to all templates. In other words, the top-level template and those defining all nested resources. The parameter defaults are defined in the **`parameter_defaults`** section of an environment file.



IMPORTANT

It is recommended to use **parameter_defaults** instead of **parameters** When creating custom environment files for your Overcloud. This is so the parameters apply to all stack templates for the Overcloud.

An example of a basic environment file:

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

parameter_defaults:
  NetworkName: my_network

parameters:
  MyIP: 192.168.0.1
```

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

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

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

6.1.3. Core Overcloud Heat Templates

The director contains a core heat template collection for the Overcloud. This collection is stored in **/usr/share/openstack-tripleo-heat-templates**.

There are many heat templates and environment files in this collection. However, the main files and directories to note in this template collection are:

- **overcloud.yaml** - This is the main template file used to create the Overcloud environment.
- **overcloud-resource-registry-puppet.yaml** - This is the main environment file used to create the Overcloud environment. It provides a set of configurations for Puppet modules stored on the Overcloud image. After the director writes the Overcloud image to each node, heat starts the Puppet configuration for each node using the resources registered in this environment file.
- **environments** - A directory that contains example environment files to apply to your Overcloud deployment.

6.2. ISOLATING NETWORKS

The director provides methods to configure isolated Overcloud networks. This means the Overcloud environment separates network traffic types into different networks, which in turn assigns network traffic to specific network interfaces or bonds. After configuring isolated networks, the director configures the

OpenStack services to use the isolated networks. If no isolated networks are configured, all services run on the Provisioning network.

This example uses separate networks for all services:

- Network 1 - Provisioning
- Network 2 - Internal API
- Network 3 - Tenant Networks
- Network 4 - Storage
- Network 5 - Storage Management
- Network 6 - Management
- Network 7 - External and Floating IP (mapped after Overcloud creation)

In this example, each Overcloud node uses two network interfaces in a bond to serve networks in tagged VLANs. The following network assignments apply to this bond:

Table 6.1. Network Subnet and VLAN Assignments

Network Type	Subnet	VLAN
Internal API	172.16.0.0/24	201
Tenant	172.17.0.0/24	202
Storage	172.18.0.0/24	203
Storage Management	172.19.0.0/24	204
Management	172.20.0.0/24	205
External / Floating IP	10.1.1.0/24	100

For more examples of network configuration, see Section 3.2, “Planning Networks”.

6.2.1. Creating Custom Interface Templates

The Overcloud network configuration requires a set of the network interface templates. You customize these templates to configure the node interfaces on a per role basis. These templates are standard Heat templates in YAML format (see Section 6.1.1, “Heat Templates”). The director contains a set of example templates to get you started:

- **`/usr/share/openstack-tripleo-heat-templates/network/config/single-nic-vlans`** - Directory containing templates for single NIC with VLANs configuration on a per role basis.
- **`/usr/share/openstack-tripleo-heat-templates/network/config/bond-with-vlans`** - Directory containing templates for bonded NIC configuration on a per role basis.

- **/usr/share/openstack-tripleo-heat-templates/network/config/multiple-nics** - Directory containing templates for multiple NIC configuration using one NIC per role.
- **/usr/share/openstack-tripleo-heat-templates/network/config/single-nic-linux-bridge-vlans** - Directory containing templates for single NIC with VLANs configuration on a per role basis and using a Linux bridge instead of an Open vSwitch bridge.

For this example, use the default bonded NIC example configuration as a basis. Copy the version located at **/usr/share/openstack-tripleo-heat-templates/network/config/bond-with-vlans**.

```
$ cp -r /usr/share/openstack-tripleo-heat-templates/network/config/bond-
with-vlans ~/templates/nic-configs
```

This creates a local set of heat templates that define a bonded network interface configuration for each role. Each template contains the standard **parameters**, **resources**, and **output** sections. For this example, you would only edit the **resources** section. Each **resources** section begins with the following:

```
resources:
OsNetConfigImpl:
  type: OS::Heat::StructuredConfig
  properties:
    group: os-apply-config
    config:
      os_net_config:
        network_config:
```

This creates a request for the **os-apply-config** command and **os-net-config** subcommand to configure the network properties for a node. The **network_config** section contains your custom interface configuration arranged in a sequence based on type, which includes the following:

interface

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

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

vlan

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

```
          - type: vlan
            vlan_id: {get_param: ExternalNetworkVlanID}
            addresses:
              - ip_netmask: {get_param: ExternalIpSubnet}
```

ovs_bond

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

```
          - type: ovs_bond
```

```
            name: bond1
            members:
            - type: interface
              name: nic2
            - type: interface
              name: nic3
```

ovs_bridge

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

```
          - type: ovs_bridge
            name: {get_input: bridge_name}
            members:
              - type: ovs_bond
                name: bond1
                members:
                  - type: interface
                    name: nic2
                    primary: true
                  - type: interface
                    name: nic3
              - type: vlan
                device: bond1
                vlan_id: {get_param: ExternalNetworkVlanID}
                addresses:
                  - ip_netmask: {get_param: ExternalIpSubnet}
```

linux_bond

Defines a Linux bond that joins two or more **`interfaces`** together. This helps with redundancy and increases bandwidth. Make sure to include the kernel-based bonding options in the **`bonding_options`** parameter. For more information on Linux bonding options, see 4.5.1. Bonding Module Directives in the Red Hat Enterprise Linux 7 Networking Guide.

```
            - type: linux_bond
              name: bond1
              members:
              - type: interface
                name: nic2
              - type: interface
                name: nic3
              bonding_options: "mode=802.3ad"
```

linux_bridge

Defines a Linux bridge, which connects multiple **`interface`**, **`linux_bond`** and **`vlan`** objects together.

```
            - type: linux_bridge
              name: bridge1
              addresses:
                - ip_netmask:
                    list_join:
                      - '/'
```

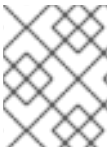
```
                    - - {get_param: ControlPlaneIp}
                      - {get_param: ControlPlaneSubnetCidr}
              members:
                - type: interface
                  name: nic1
                  primary: true
            - type: vlan
              vlan_id: {get_param: ExternalNetworkVlanID}
              device: bridge1
              addresses:
                - ip_netmask: {get_param: ExternalIpSubnet}
              routes:
                - ip_netmask: 0.0.0.0/0
                  default: true
                  next_hop: {get_param: ExternalInterfaceDefaultRoute}
```

See Appendix E, *Network Interface Parameters* for a full list of parameters for each of these items.

For this example, you use the default bonded interface configuration. For example, the **/home/stack/templates/nic-configs/controller.yaml** template uses the following **network_config**:

```
resources:
  OsNetConfigImpl:
    type: OS::Heat::StructuredConfig
    properties:
      group: os-apply-config
      config:
        os_net_config:
          network_config:
            - type: interface
              name: nic1
              use_dhcp: false
              addresses:
                - ip_netmask:
                    list_join:
                      - '/'
                      - - {get_param: ControlPlaneIp}
                        - {get_param: ControlPlaneSubnetCidr}
              routes:
                - ip_netmask: 169.254.169.254/32
                  next_hop: {get_param: EC2MetadataIp}
            - type: ovs_bridge
              name: {get_input: bridge_name}
              dns_servers: {get_param: DnsServers}
              members:
                - type: ovs_bond
                  name: bond1
                  ovs_options: {get_param: BondInterfaceOvsOptions}
                  members:
                    - type: interface
                      name: nic2
                      primary: true
                    - type: interface
                      name: nic3
```

```
                - type: vlan
                  device: bond1
                  vlan_id: {get_param: ExternalNetworkVlanID}
                  addresses:
                    - ip_netmask: {get_param: ExternalIpSubnet}
                  routes:
                    - default: true
                      next_hop: {get_param:
ExternalInterfaceDefaultRoute}
                - type: vlan
                  device: bond1
                  vlan_id: {get_param: InternalApiNetworkVlanID}
                  addresses:
                    - ip_netmask: {get_param: InternalApiIpSubnet}
                - type: vlan
                  device: bond1
                  vlan_id: {get_param: StorageNetworkVlanID}
                  addresses:
                    - ip_netmask: {get_param: StorageIpSubnet}
                - type: vlan
                  device: bond1
                  vlan_id: {get_param: StorageMgmtNetworkVlanID}
                  addresses:
                    - ip_netmask: {get_param: StorageMgmtIpSubnet}
                - type: vlan
                  device: bond1
                  vlan_id: {get_param: TenantNetworkVlanID}
                  addresses:
                    - ip_netmask: {get_param: TenantIpSubnet}
                - type: vlan
                  device: bond1
                  vlan_id: {get_param: ManagementNetworkVlanID}
                  addresses:
                    - ip_netmask: {get_param: ManagementIpSubnet}
```



NOTE

The Management network section is commented in the network interface Heat templates. Uncomment this section to enable the Management network.

This template defines a bridge (usually the external bridge named **br-ex**) and creates a bonded interface called **bond1** from two numbered interfaces: **nic2** and **nic3**. The bridge also contains a number of tagged VLAN devices, which use **bond1** as a parent device. The template also include an interface that connects back to the director (**nic1**).

For more examples of network interface templates, see Appendix F, *Network Interface Template Examples*.

Note that a lot of these parameters use the **get_param** function. You would define these in an environment file you create specifically for your networks.

IMPORTANT

Unused interfaces can cause unwanted default routes and network loops. For example, your template might contain a network interface (**nic4**) that does not use any IP assignments for OpenStack services but still uses DHCP and/or a default route. To avoid network conflicts, remove any unused interfaces from **ovs_bridge** devices and disable the DHCP and default route settings:

```
- type: interface
  name: nic4
  use_dhcp: false
  defroute: false
```

6.2.2. Creating a Network Environment File

The network environment file is a Heat environment file that describes the Overcloud's network environment and points to the network interface configuration templates from the previous section. You can define the subnets and VLANs for your network along with IP address ranges. You can then customize these values for the local environment.

The director contains a set of example environment files to get you started. Each environment file corresponds to the example network interface files in **/usr/share/openstack-tripleo-heat-templates/network/config/**:

- **/usr/share/openstack-tripleo-heat-templates/environments/net-single-nic-with-vlans.yaml** - Example environment file for single NIC with VLANs configuration in the **single-nic-vlans**) network interface directory. Environment files for disabling the External network (**net-single-nic-with-vlans-no-external.yaml**) or enabling IPv6 (**net-single-nic-with-vlans-v6.yaml**) are also available.

- **/usr/share/openstack-tripleo-heat-templates/environments/net-bond-with-vlans.yaml** - Example environment file for bonded NIC configuration in the **bond-with-vlans** network interface directory. Environment files for disabling the External network (**net-bond-with-vlans-no-external.yaml**) or enabling IPv6 (**net-bond-with-vlans-v6.yaml**) are also available.

- **/usr/share/openstack-tripleo-heat-templates/environments/net-multiple-nics.yaml** - Example environment file for a multiple NIC configuration in the **multiple-nics** network interface directory. An environment file for enabling IPv6 (**net-multiple-nics-v6.yaml**) is also available.

- **/usr/share/openstack-tripleo-heat-templates/environments/net-single-nic-linux-bridge-with-vlans.yaml** - Example environment file for single NIC with VLANs configuration using a Linux bridge instead of an Open vSwitch bridge, which uses the the **single-nic-linux-bridge-vlans** network interface directory.

This scenario uses a modified version of the **/usr/share/openstack-tripleo-heat-templates/environments/net-bond-with-vlans.yaml** file. Copy this file to the stack user's **templates** directory.

```
$ cp /usr/share/openstack-tripleo-heat-templates/environments/net-bond-
with-vlans.yaml /home/stack/templates/network-environment.yaml
```

Modify the environment file so that it contains parameters related to your environment, especially:

- The **resource_registry** section should contain modified links to the custom network interface templates for each node role. See Section 6.1.2, “Environment Files”.
- The **parameter_defaults** section should contain a list of parameters that define the network options for each network type. For a full reference of these options, see Appendix G, *Network Environment Options*.

For example:

```
resource_registry:
  OS::TripleO::BlockStorage::Net::SoftwareConfig:
/home/stack/templates/nic-configs/cinder-storage.yaml
  OS::TripleO::Compute::Net::SoftwareConfig: /home/stack/templates/nic-
configs/compute.yaml
  OS::TripleO::Controller::Net::SoftwareConfig: /home/stack/templates/nic-
configs/controller.yaml
  OS::TripleO::ObjectStorage::Net::SoftwareConfig:
/home/stack/templates/nic-configs/swift-storage.yaml
  OS::TripleO::CephStorage::Net::SoftwareConfig:
/home/stack/templates/nic-configs/ceph-storage.yaml

parameter_defaults:
  InternalApiNetCidr: 172.16.0.0/24
  TenantNetCidr: 172.17.0.0/24
  StorageNetCidr: 172.18.0.0/24
  StorageMgmtNetCidr: 172.19.0.0/24
  ManagementNetCidr: 172.20.0.0/24
  ExternalNetCidr: 10.1.1.0/24
  InternalApiAllocationPools: [{'start': '172.16.0.10', 'end':
'172.16.0.200'}]
  TenantAllocationPools: [{'start': '172.17.0.10', 'end': '172.17.0.200'}]
  StorageAllocationPools: [{'start': '172.18.0.10', 'end':
'172.18.0.200'}]
  StorageMgmtAllocationPools: [{'start': '172.19.0.10', 'end':
'172.19.0.200'}]
  ManagementAllocationPools: [{'start': '172.20.0.10', 'end':
'172.20.0.200'}]
  # Leave room for floating IPs in the External allocation pool
  ExternalAllocationPools: [{'start': '10.1.1.10', 'end': '10.1.1.50'}]
  # Set to the router gateway on the external network
  ExternalInterfaceDefaultRoute: 10.1.1.1
  # Gateway router for the provisioning network (or Undercloud IP)
  ControlPlaneDefaultRoute: 192.0.2.254
  # The IP address of the EC2 metadata server. Generally the IP of the
Undercloud
  EC2MetadataIp: 192.0.2.1
  # Define the DNS servers (maximum 2) for the overcloud nodes
  DnsServers: ["8.8.8.8","8.8.4.4"]
  InternalApiNetworkVlanID: 201
  StorageNetworkVlanID: 202
  StorageMgmtNetworkVlanID: 203
  TenantNetworkVlanID: 204
  ManagementNetworkVlanID: 205
  ExternalNetworkVlanID: 100
  # Set to "br-ex" if using floating IPs on native VLAN on bridge br-ex
  NeutronExternalNetworkBridge: "''"
```

```
  # Customize bonding options if required
  BondInterfaceOvsOptions:
    "bond_mode=balance-slb"
```

This scenario defines options for each network. All network types use an individual VLAN and subnet used for assigning IP addresses to hosts and virtual IPs. In the example above, the allocation pool for the Internal API network starts at 172.16.0.10 and continues to 172.16.0.200 using VLAN 201. This results in static and virtual IPs assigned starting at 172.16.0.10 and upwards to 172.16.0.200 while using VLAN 201 in your environment.

The External network hosts the Horizon dashboard and Public API. If using the External network for both cloud administration and floating IPs, make sure there is room for a pool of IPs to use as floating IPs for VM instances. In this example, you only have IPs from 10.1.1.10 to 10.1.1.50 assigned to the External network, which leaves IP addresses from 10.1.1.51 and above free to use for Floating IP addresses. Alternately, place the Floating IP network on a separate VLAN and configure the Overcloud after creation to use it.

The **`BondInterfaceOvsOptions`** option provides options for our bonded interface using **`nic2`** and **`nic3`**. For more information on bonding options, see Appendix H, *Open vSwitch Bonding Options*.

IMPORTANT

Changing the network configuration after creating the Overcloud can cause configuration problems due to the availability of resources. For example, if a user changes a subnet range for a network in the network isolation templates, the reconfiguration might fail due to the subnet already being in use.

6.2.3. Assigning OpenStack Services to Isolated Networks

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

For example, you can reassign the Storage Management network services to the Storage Network by modifying the **`ServiceNetMap`**:

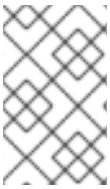
```
parameter_defaults:
  ...
  ServiceNetMap:
    NeutronTenantNetwork: tenant
    CeilometerApiNetwork: internal_api
    AodhApiNetwork: internal_api
    GnocchiApiNetwork: internal_api
    MongoDbNetwork: internal_api
    CinderApiNetwork: internal_api
    CinderIscsiNetwork: storage
    GlanceApiNetwork: storage
    GlanceRegistryNetwork: internal_api
    KeystoneAdminApiNetwork: ctlplane
    KeystonePublicApiNetwork: internal_api
    NeutronApiNetwork: internal_api
```

```
    HeatApiNetwork: internal_api
    NovaApiNetwork: internal_api
    NovaMetadataNetwork: internal_api
    NovaVncProxyNetwork: internal_api
    NovaLibvirtNetwork: internal_api
    SwiftMgmtNetwork: storage         # Change from 'storage_mgmt'
    SwiftProxyNetwork: storage
    SaharaApiNetwork: internal_api
    HorizonNetwork: internal_api
    MemcachedNetwork: internal_api
    RabbitMqNetwork: internal_api
    RedisNetwork: internal_api
    MysqlNetwork: internal_api
    CephClusterNetwork: storage       # Change from 'storage_mgmt'
    CephPublicNetwork: storage
    ControllerHostnameResolveNetwork: internal_api
    ComputeHostnameResolveNetwork: internal_api
    BlockStorageHostnameResolveNetwork: internal_api
    ObjectStorageHostnameResolveNetwork: internal_api
    CephStorageHostnameResolveNetwork: storage
    NovaColdMigrationNetwork: ctlplane
    NovaLibvirtNetwork: internal_api
  ...
```

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

6.2.4. Selecting Networks to Deploy

The settings in the **resource_registry** section of the environment file for networks and ports do not ordinarily need to be changed. The list of networks can be changed if only a subset of the networks are desired.



NOTE

When specifying custom networks and ports, do not include the **environments/network-isolation.yaml** on the deployment command line. Instead, specify all the networks and ports in the network environment file.

In order to use isolated networks, the servers must have IP addresses on each network. You can use neutron in the Undercloud to manage IP addresses on the isolated networks, so you will need to enable neutron port creation for each network. You can override the resource registry in your environment file.

First, this is the complete set of networks and ports that can be deployed:

```
resource_registry:
  # This section is usually not modified, if in doubt stick to the
defaults
  # TripleO overcloud networks
  OS::TripleO::Network::External: /usr/share/openstack-tripleo-heat-
templates/network/external.yaml
  OS::TripleO::Network::InternalApi: /usr/share/openstack-tripleo-heat-
templates/network/internal_api.yaml
  OS::TripleO::Network::StorageMgmt: /usr/share/openstack-tripleo-heat-
```

```
templates/network/storage_mgmt.yaml
  OS::TripleO::Network::Storage: /usr/share/openstack-tripleo-heat-
templates/network/storage.yaml
  OS::TripleO::Network::Tenant: /usr/share/openstack-tripleo-heat-
templates/network/tenant.yaml
  OS::TripleO::Network::Management: /usr/share/openstack-tripleo-heat-
templates/network/management.yaml

  # Port assignments for the VIPs
  OS::TripleO::Network::Ports::ExternalVipPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/external.yaml
  OS::TripleO::Network::Ports::InternalApiVipPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/internal_api.yaml
  OS::TripleO::Network::Ports::StorageVipPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/storage.yaml
  OS::TripleO::Network::Ports::StorageMgmtVipPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/storage_mgmt.yaml
  OS::TripleO::Network::Ports::TenantVipPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/tenant.yaml
  OS::TripleO::Network::Ports::ManagementVipPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/management.yaml
  OS::TripleO::Network::Ports::RedisVipPort: /usr/share/openstack-tripleo-
heat-templates/network/ports/vip.yaml

  # Port assignments for the controller role
  OS::TripleO::Controller::Ports::ExternalPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/external.yaml
  OS::TripleO::Controller::Ports::InternalApiPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/internal_api.yaml
  OS::TripleO::Controller::Ports::StoragePort: /usr/share/openstack-
tripleo-heat-templates/network/ports/storage.yaml
  OS::TripleO::Controller::Ports::StorageMgmtPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/storage_mgmt.yaml
  OS::TripleO::Controller::Ports::TenantPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/tenant.yaml
  OS::TripleO::Controller::Ports::ManagementPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/management.yaml

  # Port assignments for the compute role
  OS::TripleO::Compute::Ports::InternalApiPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/internal_api.yaml
  OS::TripleO::Compute::Ports::StoragePort: /usr/share/openstack-tripleo-
heat-templates/network/ports/storage.yaml
  OS::TripleO::Compute::Ports::TenantPort: /usr/share/openstack-tripleo-
heat-templates/network/ports/tenant.yaml
  OS::TripleO::Compute::Ports::ManagementPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/management.yaml

  # Port assignments for the ceph storage role
  OS::TripleO::CephStorage::Ports::StoragePort: /usr/share/openstack-
tripleo-heat-templates/network/ports/storage.yaml
  OS::TripleO::CephStorage::Ports::StorageMgmtPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/storage_mgmt.yaml
  OS::TripleO::CephStorage::Ports::ManagementPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/management.yaml
```

```
  # Port assignments for the swift storage role
  OS::TripleO::SwiftStorage::Ports::InternalApiPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/internal_api.yaml
  OS::TripleO::SwiftStorage::Ports::StoragePort: /usr/share/openstack-
tripleo-heat-templates/network/ports/storage.yaml
  OS::TripleO::SwiftStorage::Ports::StorageMgmtPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/storage_mgmt.yaml
  OS::TripleO::SwiftStorage::Ports::ManagementPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/management.yaml

  # Port assignments for the block storage role
  OS::TripleO::BlockStorage::Ports::InternalApiPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/internal_api.yaml
  OS::TripleO::BlockStorage::Ports::StoragePort: /usr/share/openstack-
tripleo-heat-templates/network/ports/storage.yaml
  OS::TripleO::BlockStorage::Ports::StorageMgmtPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/storage_mgmt.yaml
  OS::TripleO::BlockStorage::Ports::ManagementPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/management.yaml
```

The first section of this file has the resource registry declaration for the **`OS::TripleO::Network::*`** resources. By default these resources point at a **`noop.yaml`** file that does not create any networks. By pointing these resources at the YAML files for each network, you enable the creation of these networks.

The next several sections create the IP addresses for the nodes in each role. The controller nodes have IPs on each network. The compute and storage nodes each have IPs on a subset of the networks.

To deploy without one of the pre-configured networks, disable the network definition and the corresponding port definition for the role. For example, all references to **`storage_mgmt.yaml`** could be replaced with **`noop.yaml`**:

```
resource_registry:
  # This section is usually not modified, if in doubt stick to the
defaults
  # TripleO overcloud networks
  OS::TripleO::Network::External: /usr/share/openstack-tripleo-heat-
templates/network/external.yaml
  OS::TripleO::Network::InternalApi: /usr/share/openstack-tripleo-heat-
templates/network/internal_api.yaml
  OS::TripleO::Network::StorageMgmt: /usr/share/openstack-tripleo-heat-
templates/network/noop.yaml
  OS::TripleO::Network::Storage: /usr/share/openstack-tripleo-heat-
templates/network/storage.yaml
  OS::TripleO::Network::Tenant: /usr/share/openstack-tripleo-heat-
templates/network/tenant.yaml

  # Port assignments for the VIPs
  OS::TripleO::Network::Ports::ExternalVipPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/external.yaml
  OS::TripleO::Network::Ports::InternalApiVipPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/internal_api.yaml
  OS::TripleO::Network::Ports::StorageVipPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/storage.yaml
  OS::TripleO::Network::Ports::StorageMgmtVipPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/noop.yaml
  OS::TripleO::Network::Ports::TenantVipPort: /usr/share/openstack-
```

```
tripleo-heat-templates/network/ports/tenant.yaml
  OS::TripleO::Network::Ports::RedisVipPort: /usr/share/openstack-tripleo-
heat-templates/network/ports/vip.yaml

  # Port assignments for the controller role
  OS::TripleO::Controller::Ports::ExternalPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/external.yaml
  OS::TripleO::Controller::Ports::InternalApiPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/internal_api.yaml
  OS::TripleO::Controller::Ports::StoragePort: /usr/share/openstack-
tripleo-heat-templates/network/ports/storage.yaml
  OS::TripleO::Controller::Ports::StorageMgmtPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/noop.yaml
  OS::TripleO::Controller::Ports::TenantPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/tenant.yaml

  # Port assignments for the compute role
  OS::TripleO::Compute::Ports::InternalApiPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/internal_api.yaml
  OS::TripleO::Compute::Ports::StoragePort: /usr/share/openstack-tripleo-
heat-templates/network/ports/storage.yaml
  OS::TripleO::Compute::Ports::TenantPort: /usr/share/openstack-tripleo-
heat-templates/network/ports/tenant.yaml

  # Port assignments for the ceph storage role
  OS::TripleO::CephStorage::Ports::StoragePort: /usr/share/openstack-
tripleo-heat-templates/network/ports/storage.yaml
  OS::TripleO::CephStorage::Ports::StorageMgmtPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/noop.yaml

  # Port assignments for the swift storage role
  OS::TripleO::SwiftStorage::Ports::InternalApiPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/internal_api.yaml
  OS::TripleO::SwiftStorage::Ports::StoragePort: /usr/share/openstack-
tripleo-heat-templates/network/ports/storage.yaml
  OS::TripleO::SwiftStorage::Ports::StorageMgmtPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/noop.yaml

  # Port assignments for the block storage role
  OS::TripleO::BlockStorage::Ports::InternalApiPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/internal_api.yaml
  OS::TripleO::BlockStorage::Ports::StoragePort: /usr/share/openstack-
tripleo-heat-templates/network/ports/storage.yaml
  OS::TripleO::BlockStorage::Ports::StorageMgmtPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/noop.yaml

parameter_defaults:
  ServiceNetMap:
    NeutronTenantNetwork: tenant
    CeilometerApiNetwork: internal_api
    AodhApiNetwork: internal_api
    GnocchiApiNetwork: internal_api
    MongoDbNetwork: internal_api
    CinderApiNetwork: internal_api
    CinderIscsiNetwork: storage
    GlanceApiNetwork: storage
```

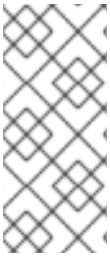
```
    GlanceRegistryNetwork: internal_api
    KeystoneAdminApiNetwork: ctlplane # Admin connection for Undercloud
    KeystonePublicApiNetwork: internal_api
    NeutronApiNetwork: internal_api
    HeatApiNetwork: internal_api
    NovaApiNetwork: internal_api
    NovaMetadataNetwork: internal_api
    NovaVncProxyNetwork: internal_api
    SwiftMgmtNetwork: storage # Changed from storage_mgmt
    SwiftProxyNetwork: storage
    SaharaApiNetwork: internal_api
    HorizonNetwork: internal_api
    MemcachedNetwork: internal_api
    RabbitMqNetwork: internal_api
    RedisNetwork: internal_api
    MysqlNetwork: internal_api
    CephClusterNetwork: storage # Changed from storage_mgmt
    CephPublicNetwork: storage
    ControllerHostnameResolveNetwork: internal_api
    ComputeHostnameResolveNetwork: internal_api
    BlockStorageHostnameResolveNetwork: internal_api
    ObjectStorageHostnameResolveNetwork: internal_api
    CephStorageHostnameResolveNetwork: storage
```

By using **noop.yaml**, no network or ports are created, so the services on the Storage Management network would default to the Provisioning network. This can be changed in the **ServiceNetMap** in order to move the Storage Management services to another network, such as the Storage network.

6.3. CONTROLLING NODE PLACEMENT

The default behavior for the director is to randomly select nodes for each role, usually based on their profile tag. However, the director provides the ability to define specific node placement. This is a useful method to:

- Assign specific node IDs e.g. **controller-0**, **controller-1**, etc
- Assign custom hostnames
- Assign specific IP addresses
- Assign specific Virtual IP addresses



NOTE

Manually setting predictable IP addresses, virtual IP addresses, and ports for a network alleviates the need for allocation pools. However, it is recommended to retain allocation pools for each network to ease with scaling new nodes. Make sure that any statically defined IP addresses fall outside the allocation pools. For more information on setting allocation pools, see Section 6.2.2, “Creating a Network Environment File”.

6.3.1. Assigning Specific Node IDs

This procedure assigns node ID to specific nodes. Examples of node IDs include **controller-0**, **controller-1**, **compute-0**, **compute-1**, and so forth.

The first step is to assign the ID as a per-node capability that the Nova scheduler matches on deployment. For example:

```
ironic node-update <id> replace properties/capabilities='node:controller-
0,boot_option:local'
```

This assigns the capability **node:controller-0** to the node. Repeat this pattern using a unique continuous index, starting from 0, for all nodes. Make sure all nodes for a given role (Controller, Compute, or each of the storage roles) are tagged in the same way or else the Nova scheduler will not match the capabilities correctly.

The next step is to create a Heat environment file (for example, **scheduler_hints_env.yaml**) that uses scheduler hints to match the capabilities for each node. For example:

```
parameter_defaults:
  ControllerSchedulerHints:
    'capabilities:node': 'controller-%index%'
```

To use these scheduler hints, include the **scheduler_hints_env.yaml** environment file with the **overcloud deploy command** during Overcloud creation.

The same approach is possible for each role via these parameters:

- **ControllerSchedulerHints** for Controller nodes.
- **NovaComputeSchedulerHints** for Compute nodes.
- **BlockStorageSchedulerHints** for Block Storage nodes.
- **ObjectStorageSchedulerHints** for Object Storage nodes.
- **CephStorageSchedulerHints** for Ceph Storage nodes.

NOTE

Node placement takes priority over profile matching. To avoid scheduling failures, use the default **baremetal** flavor for deployment and not the flavors designed for profile matching (**compute**, **control**, etc). For example:

```
$ openstack overcloud deploy ... --control-flavor baremetal --
compute-flavor baremetal ...
```

6.3.2. Assigning Custom Hostnames

In combination with the node ID configuration in Section 6.3.1, “Assigning Specific Node IDs”, the director can also assign a specific custom hostname to each node. This is useful when you need to define where a system is located (e.g. **rack2-row12**), match an inventory identifier, or other situations where a custom hostname is desired.

To customize node hostnames, use the **HostnameMap** parameter in an environment file, such as the **scheduler_hints_env.yaml** file from Section 6.3.1, “Assigning Specific Node IDs”. For example:

```
parameter_defaults:
```

```
  ControllerSchedulerHints:
    'capabilities:node': 'controller-%index%'
  NovaComputeSchedulerHints:
    'capabilities:node': 'compute-%index%'
  HostnameMap:
    overcloud-controller-0: overcloud-controller-prod-123-0
    overcloud-controller-1: overcloud-controller-prod-456-0
    overcloud-controller-2: overcloud-controller-prod-789-0
    overcloud-compute-0: overcloud-compute-prod-abc-0
```

Define the **HostnameMap** in the **parameter_defaults** section, and set each mapping as the original hostname that Heat defines using **HostnameFormat** parameters (e.g. **overcloud-controller-0**) and the second value is the desired custom hostname for that node (e.g. **overcloud-controller-prod-123-0**).

Using this method in combination with the node ID placement ensures each node has a custom hostname.

6.3.3. Assigning Predictable IPs

For further control over the resulting environment, the director can assign Overcloud nodes with specific IPs on each network as well. Use the **environments/ips-from-pool-all.yaml** environment file in the core Heat template collection. Copy this file to the **stack** user's **templates** directory.

```
$ cp /usr/share/openstack-tripleo-heat-templates/environments/ips-from-
pool-all.yaml ~/templates/.
```

There are two major sections in the **ips-from-pool-all.yaml** file.

The first is a set of **resource_registry** references that override the defaults. These tell the director to use a specific IP for a given port on a node type. Modify each resource to use the absolute path of its respective template. For example:

```
  OS::TripleO::Controller::Ports::ExternalPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/external_from_pool.yaml
  OS::TripleO::Controller::Ports::InternalApiPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/internal_api_from_pool.yaml
  OS::TripleO::Controller::Ports::StoragePort: /usr/share/openstack-
tripleo-heat-templates/network/ports/storage_from_pool.yaml
  OS::TripleO::Controller::Ports::StorageMgmtPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/storage_mgmt_from_pool.yaml
  OS::TripleO::Controller::Ports::TenantPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/tenant_from_pool.yaml
```

The default configuration sets all networks on all node types to use pre-assigned IPs. To allow a particular network or node type to use default IP assignment instead, simply remove the **resource_registry** entries related to that node type or network from the environment file.

The second section is parameter_defaults, where the actual IP addresses are assigned. Each node type has an associated parameter:

- **ControllerIPs** for Controller nodes.
- **NovaComputeIPs** for Compute nodes.

- **`CephStorageIPs`** for Ceph Storage nodes.
- **`BlockStorageIPs`** for Block Storage nodes.
- **`SwiftStorageIPs`** for Object Storage nodes.

Each parameter is a map of network names to a list of addresses. Each network type must have at least as many addresses as there will be nodes on that network. The director assigns addresses in order. The first node of each type receives the first address on each respective list, the second node receives the second address on each respective lists, and so forth.

For example, if an Overcloud will contain three Ceph Storage nodes, the CephStorageIPs parameter might look like:

```
CephStorageIPs:
  storage:
  - 172.16.1.100
  - 172.16.1.101
  - 172.16.1.102
  storage_mgmt:
  - 172.16.3.100
  - 172.16.3.101
  - 172.16.3.102
```

The first Ceph Storage node receives two addresses: 172.16.1.100 and 172.16.3.100. The second receives 172.16.1.101 and 172.16.3.101, and the third receives 172.16.1.102 and 172.16.3.102. The same pattern applies to the other node types.

Make sure the chosen IP addresses fall outside the allocation pools for each network defined in your network environment file (see Section 6.2.2, “Creating a Network Environment File”). For example, make sure the **`internal_api`** assignments fall outside of the **`InternalApiAllocationPools`** range. This avoids conflicts with any IPs chosen automatically. Likewise, make sure the IP assignments do not conflict with the VIP configuration, either for standard predictable VIP placement (see Section 6.3.4, “Assigning Predictable Virtual IPs”) or external load balancing (see Section 6.5, “Configuring External Load Balancing”).

To apply this configuration during a deployment, include the environment file with the **`openstack overcloud deploy`** command. If using network isolation (see Section 6.2, “Isolating Networks”), include this file after the **`network-isolation.yaml`** file. For example:

```
$ openstack overcloud deploy --templates -e /usr/share/openstack-tripleo-
heat-templates/environments/network-isolation.yaml -e ~/templates/ips-
from-pool-all.yaml [OTHER OPTIONS]
```

6.3.4. Assigning Predictable Virtual IPs

In addition to defining predictable IP addresses for each node, the director also provides a similar ability to define predictable Virtual IPs (VIPs) for clustered services. To accomplish this, edit the network environment file from Section 6.2.2, “Creating a Network Environment File” and add the VIP parameters in the **`parameter_defaults`** section:

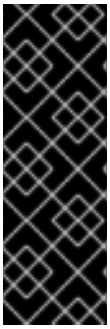
```
parameter_defaults:
  ...
  # Predictable VIPs
```

```
  ControlFixedIPs: [{'ip_address':'192.168.201.101'}]
  InternalApiVirtualFixedIPs: [{'ip_address':'172.16.0.9'}]
  PublicVirtualFixedIPs: [{'ip_address':'10.1.1.9'}]
  StorageVirtualFixedIPs: [{'ip_address':'172.18.0.9'}]
  StorageMgmtVirtualFixedIPs: [{'ip_address':'172.19.0.9'}]
  RedisVirtualFixedIPs: [{'ip_address':'172.16.0.8'}]
```

Select these IPs from outside of their respective allocation pool ranges. For example, select an IP address for **`InternalApiVirtualFixedIPs`** that is not within the **`InternalApiAllocationPools`** range.

6.4. CONFIGURING CONTAINERIZED COMPUTE NODES

The director provides an option to integrate services from OpenStack's containerization project (kolla) into the Overcloud's Compute nodes. This includes creating Compute nodes that use Red Hat Enterprise Linux Atomic Host as a base operating system and individual containers to run different OpenStack services.



IMPORTANT

Containerized Compute nodes are a Technology Preview feature. Technology Preview features are not fully supported under Red Hat Subscription Service Level Agreements (SLAs), may not be functionally complete, and are not intended for production use. However, these features provide early access to upcoming product innovations, enabling customers to test functionality and provide feedback during the development process. For more information on the support scope for features marked as technology previews, see https://access.redhat.com/support/offerings/techpreview/.

The director's core Heat template collection includes environment files to aid the configuration of containerized Compute nodes. These files include:

- **`docker.yaml`** - The main environment file for configuring containerized Compute nodes.
- **`docker-network.yaml`** - The environment file for containerized Compute nodes networking without network isolation.
- **`docker-network-isolation.yaml`** - The environment file for containerized Compute nodes using network isolation.

6.4.1. Examining the Containerized Compute Environment File (docker.yaml)

The **`docker.yaml`** file is the main environment file for the containerized Compute node configuration. It includes the entries in the **`resource_registry`**:

```
resource_registry:
  OS::TripleO::ComputePostDeployment: ../docker/compute-post.yaml
  OS::TripleO::NodeUserData:
../docker/firstboot/install_docker_agents.yaml
```

OS::TripleO::NodeUserData

Provides a Heat template that uses custom configuration on first boot. In this case, it installs the **`openstack-heat-docker-agents`** container on the Compute nodes when they first boot. This container provides a set of initialization scripts to configure the containerized Compute node and Heat

hooks to communicate with the director.

OS::TripleO::ComputePostDeployment

Provides a Heat template with a set of post-configuration resources for Compute nodes. This includes a software configuration resource that provides a set of **tags** to Puppet:

```
  ComputePuppetConfig:
    type: OS::Heat::SoftwareConfig
    properties:
      group: puppet
      options:
        enable_hiera: True
        enable_facter: False
        tags:
package,file,concat,file_line,nova_config,neutron_config,neutron_agent_o
vs,neutron_plugin_ml2
      inputs:
      - name: tripleo::packages::enable_install
        type: Boolean
        default: True
      outputs:
      - name: result
      config:
        get_file: ../puppet/manifests/overcloud_compute.pp
```

These tags define the Puppet modules to pass to the **openstack-heat-docker-agents** container.

The **docker.yaml** file includes a **parameter** called **NovaImage** that replaces the standard **overcloud-full** image with a different image (**atomic-image**) when provisioning Compute nodes. See in Section 6.4.2, “Uploading the Atomic Host Image” for instructions on uploading this new image.

The **docker.yaml** file also includes a **parameter_defaults** section that defines the Docker registry and images to use for our Compute node services. You can modify this section to use a local registry instead of the default registry.access.redhat.com. See Section 6.4.3, “Using a Local Registry” for instructions on configuring a local registry.

6.4.2. Uploading the Atomic Host Image

The director requires a copy of the Cloud Image for Red Hat Enterprise Linux 7 Atomic Host imported into its image store as **atomic-image**. This is because the Compute node requires this image for the base OS during the provisioning phase of the Overcloud creation.

Download a copy of the **Cloud Image** from the Red Hat Enterprise Linux 7 Atomic Host product page (https://access.redhat.com/downloads/content/271/ver=/rhel---7/7.2.2-2/x86_64/product-software) and save it to the **images** subdirectory in the **stack** user’s home directory.

Once the image download completes, import the image into the director as the **stack** user.

```
$ glance image-create --name atomic-image --file ~/images/rhel-atomic-
cloud-7.2-12.x86_64.qcow2 --disk-format qcow2 --container-format bare
```

This imports the image alongside the other Overcloud images.

```
$ glance image-list
+--------------------------------------+------------------------+
| ID                                   | Name                   |
+--------------------------------------+------------------------+
| 27b5bad7-f8b2-4dd8-9f69-32dfe84644cf | atomic-image           |
| 08c116c6-8913-427b-b5b0-b55c18a01888 | bm-deploy-kernel       |
| aec4c104-0146-437b-a10b-8ebc351067b9 | bm-deploy-ramdisk      |
| 9012ce83-4c63-4cd7-a976-0c972be747cd | overcloud-full         |
| 376e95df-c1c1-4f2a-b5f3-93f639eb9972 | overcloud-full-initrd  |
| 0b5773eb-4c64-4086-9298-7f28606b68af | overcloud-full-vmlinuz |
+--------------------------------------+------------------------+
```

6.4.3. Using a Local Registry

The default configuration uses Red Hat's container registry for image downloads. However, as an optional step, you can use a local registry to conserve bandwidth during the Overcloud creation process.

You can use an existing local registry or install a new one. To install a new registry, use the instructions in Chapter 2. Get Started with Docker Formatted Container Images in Getting Started with Containers.

Pull the required images into your registry:

```
$ sudo docker pull
registry.access.redhat.com/rhosp9_tech_preview/openstack-nova-
compute:latest
$ sudo docker pull
registry.access.redhat.com/rhosp9_tech_preview/openstack-data:latest
$ sudo docker pull
registry.access.redhat.com/rhosp9_tech_preview/openstack-nova-
libvirt:latest
$ sudo docker pull
registry.access.redhat.com/rhosp9_tech_preview/openstack-neutron-
openvswitch-agent:latest
$ sudo docker pull
registry.access.redhat.com/rhosp9_tech_preview/openstack-openvswitch-
vswitchd:latest
$ sudo docker pull
registry.access.redhat.com/rhosp9_tech_preview/openstack-openvswitch-db-
server:latest
$ sudo docker pull
registry.access.redhat.com/rhosp9_tech_preview/openstack-heat-docker-
agents:latest
```

After pulling the images, tag them with the proper registry host:

```
$ sudo docker tag
registry.access.redhat.com/rhosp9_tech_preview/openstack-nova-
compute:latest localhost:8787/registry.access.redhat.com/openstack-nova-
compute:latest
$ sudo docker tag
registry.access.redhat.com/rhosp9_tech_preview/openstack-data:latest
localhost:8787/registry.access.redhat.com/openstack-data:latest
$ sudo docker tag
registry.access.redhat.com/rhosp9_tech_preview/openstack-nova-
libvirt:latest localhost:8787/registry.access.redhat.com/openstack-nova-
```

```
libvirt:latest
$ sudo docker tag
registry.access.redhat.com/rhosp9_tech_preview/openstack-neutron-
openvswitch-agent:latest
localhost:8787/registry.access.redhat.com/openstack-neutron-openvswitch-
agent:latest
$ sudo docker tag
registry.access.redhat.com/rhosp9_tech_preview/openstack-openvswitch-
vswitchd:latest localhost:8787/registry.access.redhat.com/openstack-
openvswitch-vswitchd:latest
$ sudo docker tag
registry.access.redhat.com/rhosp9_tech_preview/openstack-openvswitch-db-
server:latest localhost:8787/registry.access.redhat.com/openstack-
openvswitch-db-server:latest
$ sudo docker tag
registry.access.redhat.com/rhosp9_tech_preview/openstack-heat-docker-
agents:latest localhost:8787/registry.access.redhat.com/openstack-heat-
docker-agents:latest
```

Push them to the registry:

```
$ sudo docker push localhost:8787/registry.access.redhat.com/openstack-
nova-compute:latest
$ sudo docker push localhost:8787/registry.access.redhat.com/openstack-
data:latest
$ sudo docker push localhost:8787/registry.access.redhat.com/openstack-
nova-libvirt:latest
$ sudo docker push localhost:8787/registry.access.redhat.com/openstack-
neutron-openvswitch-agent:latest
$ sudo docker push localhost:8787/registry.access.redhat.com/openstack-
openvswitch-vswitchd:latest
$ sudo docker push localhost:8787/registry.access.redhat.com/openstack-
openvswitch-db-server:latest
$ sudo docker push localhost:8787/registry.access.redhat.com/openstack-
heat-docker-agents:latest
```

Create a copy of the main **`docker.yaml`** environment file in the **`templates`** subdirectory:

```
$ cp /usr/share/openstack-tripleo-heat-templates/environments/docker.yaml
~/templates/.
```

Edit the file and modify the **`resource_registry`** to use absolute paths:

```
resource_registry:
  OS::TripleO::ComputePostDeployment: /usr/share/openstack-tripleo-heat-
templates/docker/compute-post.yaml
  OS::TripleO::NodeUserData: /usr/share/openstack-tripleo-heat-
templates/docker/firstboot/install_docker_agents.yaml
```

Set **`DockerNamespace`** in **`parameter_defaults`** to your registry URL. Also set **`DockerNamespaceIsRegistry`** to **`true`** For example:

```
parameter_defaults:
  DockerNamespace: registry.example.com:8787/registry.access.redhat.com
  DockerNamespaceIsRegistry: true
```

Your local registry now has the required docker images and the containerized Compute configuration is now set to use that registry.

6.4.4. Including Environment Files in the Overcloud Deployment

When running the Overcloud creation, include the main environment file (**`docker.yaml`**) and the network environment file (**`docker-network.yaml`**) for the containerized Compute nodes along with the **`openstack overcloud deploy`** command. For example:

```
$ openstack overcloud deploy --templates -e /usr/share/openstack-tripleo-
heat-templates/environments/docker.yaml -e /usr/share/openstack-tripleo-
heat-templates/environments/docker-network.yaml [OTHER OPTIONS] ...
```

The containerized Compute nodes also function in an Overcloud with network isolation. This also requires the main environment file along with the network isolation file (**`docker-network-isolation.yaml`**). Add these files before the network isolation files from Section 6.2, “Isolating Networks”. For example:

```
openstack overcloud deploy --templates -e /usr/share/openstack-tripleo-
heat-templates/environments/docker.yaml -e /usr/share/openstack-tripleo-
heat-templates/environments/docker-network-isolation.yaml -e
/usr/share/openstack-tripleo-heat-templates/environments/net-single-nic-
with-vlans.yaml -e /usr/share/openstack-tripleo-heat-
templates/environments/network-isolation.yaml [OTHER OPTIONS] ...
```

The director creates an Overcloud with containerized Compute nodes.

6.5. CONFIGURING EXTERNAL LOAD BALANCING

An Overcloud uses multiple Controllers together as a high availability cluster, which ensures maximum operational performance for your OpenStack services. In addition, the cluster provides load balancing for access to the OpenStack services, which evenly distributes traffic to the Controller nodes and reduces server overload for each node. It is also possible to use an external load balancer to perform this distribution. For example, an organization might use their own hardware-based load balancer to handle traffic distribution to the Controller nodes.

For more information about configuring external load balancing, see the dedicated External Load Balancing for the Overcloud guide for full instructions.

6.6. CONFIGURING IPV6 NETWORKING

As a default, the Overcloud uses Internet Protocol version 4 (IPv4) to configure the service endpoints. However, the Overcloud also supports Internet Protocol version 6 (IPv6) endpoints, which is useful for organizations that support IPv6 infrastructure. The director includes a set of environment files to help with creating IPv6-based Overclouds.

For more information about configuring IPv6 in the Overcloud, see the dedicated IPv6 Networking for the Overcloud guide for full instructions.

6.7. CONFIGURING NFS STORAGE

This section describes configuring the Overcloud to use an NFS share. The installation and configuration process is based on the modification of an existing environment file in the core Heat template collection.

The core heat template collection contains a set of environment files in **`/usr/share/openstack-tripleo-heat-templates/environments/`**. These environment templates help with custom configuration of some of the supported features in a director-created Overcloud. This includes an environment file to help configure storage. This file is located at **`/usr/share/openstack-tripleo-heat-templates/environments/storage-environment.yaml`**. Copy this file to the **`stack`** user's template directory.

```
$ cp /usr/share/openstack-tripleo-heat-templates/environments/storage-
environment.yaml ~/templates/.
```

The environment file contains some parameters to help configure different storage options for Openstack's block and image storage components, cinder and glance. In this example, you will configure the Overcloud to use an NFS share. Modify the following parameters:

CinderEnableIscsiBackend

Enables the iSCSI backend. Set to **`false`**.

CinderEnableRbdBackend

Enables the Ceph Storage backend. Set to **`false`**.

CinderEnableNfsBackend

Enables the NFS backend. Set to **`true`**.

NovaEnableRbdBackend

Enables Ceph Storage for Nova ephemeral storage. Set to **`false`**.

GlanceBackend

Define the back end to use for Glance. Set to **`file`** to use file-based storage for images. The Overcloud will save these files in a mounted NFS share for Glance.

CinderNfsMountOptions

The NFS mount options for the volume storage.

CinderNfsServers

The NFS share to mount for volume storage. For example, 192.168.122.1:/export/cinder.

GlanceFilePcmkManage

Enables Pacemaker to manage the share for image storage. If disabled, the Overcloud stores images in the Controller node's file system. Set to **`true`**.

GlanceFilePcmkFstype

Defines the file system type that Pacemaker uses for image storage. Set to **`nfs`**.

GlanceFilePcmkDevice

The NFS share to mount for image storage. For example, 192.168.122.1:/export/glance.

GlanceFilePcmkOptions

The NFS mount options for the image storage.

The environment file's options should look similar to the following:

```
parameter_defaults:
```

```
  CinderEnableIscsiBackend: false
  CinderEnableRbdBackend: false
  CinderEnableNfsBackend: true
  NovaEnableRbdBackend: false
  GlanceBackend: 'file'

  CinderNfsMountOptions: 'rw,sync'
  CinderNfsServers: '192.0.2.230:/cinder'

  GlanceFilePcmkManage: true
  GlanceFilePcmkFstype: 'nfs'
  GlanceFilePcmkDevice: '192.0.2.230:/glance'
  GlanceFilePcmkOptions:
'rw,sync,context=system_u:object_r:glance_var_lib_t:s0'
```

IMPORTANT

Include the **`context=system_u:object_r:glance_var_lib_t:s0`** in the **`GlanceFilePcmkOptions`** parameter to allow glance access to the **`/var/lib`** directory. Without this SELinux content, glance will fail to write to the mount point.

These parameters are integrated as part of the heat template collection. Setting them as such creates two NFS mount points for cinder and glance to use.

Save this file for inclusion in the Overcloud creation.

6.8. CONFIGURING CEPH STORAGE

The director provides two main methods for integrating Red Hat Ceph Storage into an Overcloud.

Creating an Overcloud with its own Ceph Storage Cluster

The director has the ability to create a Ceph Storage Cluster during the creation on the Overcloud. The director creates a set of Ceph Storage nodes that use the Ceph OSD to store the data. In addition, the director install the Ceph Monitor service on the Overcloud's Controller nodes. This means if an organization creates an Overcloud with three highly available controller nodes, the Ceph Monitor also becomes a highly available service.

Integrating a Existing Ceph Storage into an Overcloud

If you already have an existing Ceph Storage Cluster, you can integrate this during an Overcloud deployment. This means you manage and scale the cluster outside of the Overcloud configuration.

For more information about configuring Overcloud Ceph Storage, see the dedicated Red Hat Ceph Storage for the Overcloud guide for full instructions on both scenarios.

6.9. CONFIGURING THIRD PARTY STORAGE

The director include a couple of environment files to help configure third-party storage providers. This includes:

Dell Storage Center

Deploys a single Dell Storage Center back end for the Block Storage (cinder) service. The environment file is located at **`/usr/share/openstack-tripleo-heat-templates/environments/cinder-dellsc-config.yaml`**.

See the Dell Storage Center Back End Guide for full configuration information.

Dell EqualLogic

Deploys a single Dell EqualLogic back end for the Block Storage (cinder) service. The environment file is located at **`/usr/share/openstack-tripleo-heat-templates/environments/cinder-eqlx-config.yaml`**.

See the Dell EqualLogic Back End Guide for full configuration information.

NetApp Block Storage

Deploys a NetApp storage appliance as a back end for the Block Storage (cinder) service. The environment file is located at **`/usr/share/openstack-tripleo-heat-templates/environments/cinder-dellsc-config.yaml/cinder-netapp-config.yaml`**.

See the NetApp Block Storage Back End Guide for full configuration information.

6.10. ENABLING SSL/TLS ON THE OVERCLOUD

By default, the Overcloud uses unencrypted endpoints for its services; this means that the Overcloud configuration requires an additional environment file to enable SSL/TLS for its Public API endpoints.

NOTE

This process only enables SSL/TLS for Public API endpoints. The Internal and Admin APIs remain unencrypted.

This process requires network isolation to define the endpoints for the Public API. See Section 6.2, “Isolating Networks” for instruction on network isolation.

Ensure you have a private key and certificate authority created. See Appendix A, *SSL/TLS Certificate Configuration* for more information on creating a valid SSL/TLS key and certificate authority file.

6.10.1. Enabling SSL/TLS

Copy the **`enable-tls.yaml`** environment file from the Heat template collection:

```
$ cp -r /usr/share/openstack-tripleo-heat-templates/environments/enable-tls.yaml ~/templates/.
```

Edit this file and make the following changes for these parameters:

SSLCertificate

Copy the contents of the certificate file into the **`SSLCertificate`** parameter. For example:

```
parameter_defaults:
  SSLCertificate: |
    -----BEGIN CERTIFICATE-----
    MIIDgzCCAmugAwIBAgIJAKk46qw6ncJaMA0GCSqGSIb3DQEBCwUAMFgxCzAJBgNV
    ...
    sFW3S2roS4X0Af/kSSD8mlBBTFTCMBAj6rtLBKLaQbIxEpIzrgvp
    -----END CERTIFICATE-----
```

IMPORTANT

The certificate authority contents require the same indentation level for all new lines.

SSLKey

Copy the contents of the private key into the **SSLKey** parameter. For example:

```
parameter_defaults:
  ...
  SSLKey: |
    -----BEGIN RSA PRIVATE KEY-----
    MIIEowIBAAKCAQEAqVw8lnQ9RbeI1EdLN5PJP0lVO9hkJZnGP6qb6wtYUoy1bVP7
    ...
    ctlKn3rAAdyumi4JDjESAXHIKFjJNOLrBmpQyES4XpZUC7yhqPaU
    -----END RSA PRIVATE KEY-----
```

IMPORTANT

The private key contents require the same indentation level for all new lines.

EndpointMap

The **EndpointMap** contains a mapping of the services using HTTPS and HTTP communication. If using DNS for SSL communication, leave this section with the defaults. However, if using an IP address for the SSL certificate's common name (see Appendix A, *SSL/TLS Certificate Configuration*), replace all instances of **CLOUDNAME** with **IP_ADDRESS**. Use the following command to accomplish this:

```
$ sed -i 's/CLOUDNAME/IP_ADDRESS/' ~/templates/enable-tls.yaml
```

IMPORTANT

Do not substitute **IP_ADDRESS** or **CLOUDNAME** for actual values. Heat replaces these variables with the appropriate value during the Overcloud creation.

OS::TripleO::NodeTLSData

Change the resource path for **OS::TripleO::NodeTLSData:** to an absolute path:

```
resource_registry:
  OS::TripleO::NodeTLSData: /usr/share/openstack-tripleo-heat-
templates/puppet/extraconfig/tls/tls-cert-inject.yaml
```

6.10.2. Injecting a Root Certificate

If the certificate signer is not in the default trust store on the Overcloud image, you must inject the certificate authority into the Overcloud image. Copy the **inject-trust-anchor.yaml** environment file from the heat template collection:

```
$ cp -r /usr/share/openstack-tripleo-heat-templates/environments/inject-
trust-anchor.yaml ~/templates/.
```

-

Edit this file and make the following changes for these parameters:

SSLRootCertificate

Copy the contents of the root certificate authority file into the **`SSLRootCertificate`** parameter. For example:

```
parameter_defaults:
  SSLRootCertificate: |
    -----BEGIN CERTIFICATE-----
    MIIDgzCCAmugAwIBAgIJAKk46qw6ncJaMA0GCSqGSIb3DQEBCwUAMFgxCzAJBgNV
    ...
    sFW3S2roS4X0Af/kSSD8mlBBTFTCMBAj6rtLBKLaQbIxEpIzrgvp
    -----END CERTIFICATE-----
```

IMPORTANT

The certificate authority contents require the same indentation level for all new lines.

OS::TripleO::NodeTLSCAData

Change the resource path for **`OS::TripleO::NodeTLSCAData:`** to an absolute path:

```
resource_registry:
  OS::TripleO::NodeTLSCAData: /usr/share/openstack-tripleo-heat-
templates/puppet/extraconfig/tls/ca-inject.yaml
```

6.10.3. Configuring DNS Endpoints

If using a DNS hostname to access the Overcloud through SSL/TLS, create a new environment file (**`~/templates/cloudname.yaml`**) to define the hostname of the Overcloud's endpoints. Use the following parameters:

CloudName

The DNS hostname of the Overcloud endpoints.

DnsServers

A list of DNS servers to use. The configured DNS servers must contain an entry for the configured **`CloudName`** that matches the IP address of the Public API.

An example of the contents for this file:

```
parameter_defaults:
  CloudName: overcloud.example.com
  DnsServers: ["10.0.0.1"]
```

6.10.4. Adding Environment Files During Overcloud Creation

The deployment command (**`openstack overcloud deploy`**) in Chapter 7, *Creating the Overcloud* uses the **`-e`** option to add environment files. Add the environment files from this section in the following order:

- The environment file to enable SSL/TLS (**enable-tls.yaml**)
- The environment file to set the DNS hostname (**cloudname.yaml**)
- The environment file to inject the root certificate authority (**inject-trust-anchor.yaml**)

For example:

```
$ openstack overcloud deploy --templates [...] -e
/home/stack/templates/enable-tls.yaml -e ~/templates/cloudname.yaml -e
~/templates/inject-trust-anchor.yaml
```

6.11. CONFIGURING BASE PARAMETERS

The Overcloud's Heat templates contain a set of base parameters that you configure prior to running the **openstack overcloud deploy** command. Include these base parameters in the **parameter_defaults** section of an environment file and include the environment file with the **openstack overcloud deploy** command.

For a full list of base parameters for the Overcloud, see Appendix D, *Base Parameters*.

Example 1: Configuring the Timezone

Add the entry in an environment file to set your timezone to **Japan**:

```
parameter_defaults:
  TimeZone: 'Japan'
```

Example 2: Disabling Layer 3 High Availability (L3HA)

If you need to disable Layer 3 High Availability (L3HA) for OpenStack Networking, add this to an environment file:

```
parameter_defaults:
  NeutronL3HA: False
```

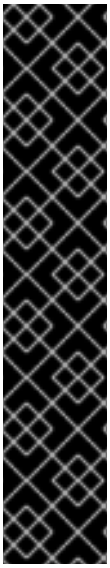
Example 3: Configuring the Telemetry Dispatcher

The OpenStack Telemetry (**ceilometer**) service includes a new component for a time series data storage (**gnocchi**). It is possible in Red Hat OpenStack Platform to switch the default Ceilometer dispatcher to use this new component instead of the standard database. You accomplish this with the **CeilometerMeterDispatcher**, which you set to either:

- **database** - Use the standard database for the Ceilometer dispatcher. This is the default option.
- **gnocchi** - Use the new time series database for Ceilometer dispatcher.

To switch to the time series database, add the following to an environment file:

```
parameter_defaults:
  CeilometerMeterDispatcher: gnocchi
```



IMPORTANT

When installing Red Hat OpenStack Platform 9 with **`CeilometerMeterDispatcher`** parameter set to **`gnocchi`**, restart the **`openstack-ceilometer-collector`** service after the install completes to make sure that the collector communicates with **gnocchi** successfully. To restart the service, use the following command:

```
$ sudo pcs resource restart openstack-ceilometer-collector-
clone
```

The step is necessary because **`keystone init`**, which registers the gnocchi user to keystone, runs at the very end of the installation process. Because of that, the gnocchi user is not in the keystone catalogue when the collector is started. Restarting the collector after the installation enables **ceilometer** to communicate with gnocchi and dispatch the data accordingly. The next release of Red Hat OpenStack Platform will solve this problem.

Example 4: Configuring RabbitMQ File Descriptor Limit

For certain configurations, you might need to increase the file descriptor limit for the RabbitMQ server. Set the **`RabbitFDLimit`** parameter to the limit you require:

```
parameter_defaults:
  RabbitFDLimit: 65536
```

6.12. REGISTERING THE OVERCLOUD

The Overcloud provides a method to register nodes to either the Red Hat Content Delivery Network, a Red Hat Satellite 5 server, or a Red Hat Satellite 6 server. You can either achieve this through environment files or the command line.

6.12.1. Method 1: Command Line

The deployment command (**`openstack overcloud deploy`**) uses a set of options to define your registration details. The table in Section 7.1, “Setting Overcloud Parameters” contains these options and their descriptions. Include these options when running the deployment command in Chapter 7, *Creating the Overcloud*. For example:

```
# openstack overcloud deploy --templates --rhel-reg --reg-method satellite
--reg-sat-url http://example.satellite.com  --reg-org MyOrg --reg-
activation-key MyKey --reg-force [...]
```

6.12.2. Method 2: Environment File

Copy the registration files from the Heat template collection:

```
$ cp -r /usr/share/openstack-tripleo-heat-
templates/extraconfig/pre_deploy/rhel-registration ~/templates/.
```

Edit the **`~/templates/rhel-registration/environment-rhel-registration.yaml`** and modify the following values to suit your registration method and details.

rhel_reg_method

Choose the registration method. Either **`portal`**, **`satellite`**, or **`disable`**.

rhel_reg_type

The type of unit to register. Leave blank to register as a **`system`**

rhel_reg_auto_attach

Automatically attach compatible subscriptions to this system. Set to **`true`** to enable.

rhel_reg_service_level

The service level to use for auto attachment.

rhel_reg_release

Use this parameter to set a release version for auto attachment. Leave blank to use the default from Red Hat Subscription Manager.

rhel_reg_pool_id

The subscription pool ID to use. Use this if not auto-attaching subscriptions.

rhel_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`**.

rhel_reg_server_url

The hostname of the subscription service to use. The default is for Customer Portal Subscription Management, subscription.rhn.redhat.com. If this option is not used, the system is registered with Customer Portal Subscription Management. The subscription server URL uses the form of https://hostname:port/prefix.

rhel_reg_base_url

Gives the hostname of the content delivery server to use to receive updates. The default is https://cdn.redhat.com. Since Satellite 6 hosts its own content, the URL must be used for systems registered with Satellite 6. The base URL for content uses the form of https://hostname:port/prefix.

rhel_reg_org

The organization to use for registration.

rhel_reg_environment

The environment to use within the chosen organization.

rhel_reg_repos

A comma-separated list of repositories to enable. See Section 2.5, "Repository Requirements" for repositories to enable.

rhel_reg_activation_key

The activation key to use for registration.

rhel_reg_user; rhel_reg_password

The username and password for registration. If possible, use activation keys for registration.

rhel_reg_machine_name

The machine name. Leave this as blank to use the hostname of the node.

rhel_reg_force

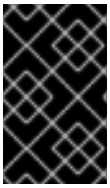
Set to **`true`** to force your registration options. For example, when re-registering nodes.

rhel_reg_sat_repo

The repository containing Red Hat Satellite 6's management tools, such as **katello-agent**. Check the correct repository name corresponds to your Red Hat Satellite version and check that the repository is synchronized on the Satellite server. For example, **rhel-7-server-satellite-tools-6.2-rpms** corresponds to Red Hat Satellite 6.2.

The deployment command (**openstack overcloud deploy**) in Chapter 7, *Creating the Overcloud* uses the **-e** option to add environment files. Add both **~/templates/rhel-registration/environment-rhel-registration.yaml** and **~/templates/rhel-registration/rhel-registration-resource-registry.yaml**. For example:

```
$ openstack overcloud deploy --templates [...] -e
/home/stack/templates/rhel-registration/environment-rhel-registration.yaml
-e /home/stack/templates/rhel-registration/rhel-registration-resource-
registry.yaml
```



IMPORTANT

Registration is set as the **OS::TripleO::NodeExtraConfig** Heat resource. This means you can only use this resource for registration. See Section 6.14, "Customizing Overcloud Pre-Configuration" for more information.

6.13. CUSTOMIZING CONFIGURATION ON FIRST BOOT

The director provides a mechanism to perform configuration on all nodes upon the initial creation of the Overcloud. The director achieves this through **cloud-init**, which you can call using the **OS::TripleO::NodeUserData** resource type.

In this example, you will update the nameserver with a custom IP address on all nodes. You must first create a basic heat template (**/home/stack/templates/nameserver.yaml**) that runs a script to append each node's **resolv.conf** with a specific nameserver. You can use the **OS::TripleO::MultipartMime** resource type to send the configuration script.

```
heat_template_version: 2014-10-16

description: >
  Extra hostname configuration

resources:
  userdata:
    type: OS::Heat::MultipartMime
    properties:
      parts:
      - config: {get_resource: nameserver_config}

  nameserver_config:
    type: OS::Heat::SoftwareConfig
    properties:
      config: |
        #!/bin/bash
        echo "nameserver 192.168.1.1" >> /etc/resolv.conf

outputs:
  OS::stack_id:
    value: {get_resource: userdata}
```

Next, create an environment file (**/home/stack/templates/firstboot.yaml**) that registers your heat template as the **OS::TripleO::NodeUserData** resource type.

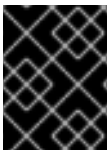
```
resource_registry:
  OS::TripleO::NodeUserData: /home/stack/templates/nameserver.yaml
```

To add the first boot configuration, add the environment file to the stack when first creating the Overcloud. For example:

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

The **-e** applies the environment file to the Overcloud stack.

This adds the configuration to all nodes when they are first created and boot for the first time. Subsequent inclusions of these templates, such as updating the Overcloud stack, does not run these scripts.



IMPORTANT

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

6.14. CUSTOMIZING OVERCLOUD PRE-CONFIGURATION

The Overcloud uses Puppet for the core configuration of OpenStack components. The director provides a set of resources to provide custom configuration after the first boot completes and before the core configuration begins. These resources include:

OS::TripleO::ControllerExtraConfigPre

Additional configuration applied to Controller nodes before the core Puppet configuration.

OS::TripleO::ComputeExtraConfigPre

Additional configuration applied to Compute nodes before the core Puppet configuration.

OS::TripleO::CephStorageExtraConfigPre

Additional configuration applied to CephStorage nodes before the core Puppet configuration.

OS::TripleO::NodeExtraConfig

Additional configuration applied to all nodes roles before the core Puppet configuration.

In this example, you first create a basic heat template (**/home/stack/templates/nameserver.yaml**) that runs a script to append each node's **resolv.conf** with a variable nameserver.

```
heat_template_version: 2014-10-16

description: >
  Extra hostname configuration

parameters:
  server:
    type: string
```

```
  nameserver_ip:
    type: string
  DeployIdentifier:
    type: string

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

  ExtraPreDeployment:
    type: OS::Heat::SoftwareDeployment
    properties:
      config: {get_resource: ExtraPreConfig}
      server: {get_param: server}
      actions: ['CREATE','UPDATE']
      input_values:
        deploy_identifier: {get_param: DeployIdentifier}

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

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

ExtraPreConfig

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

ExtraPreDeployment

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

- The **server** parameter is provided by the parent template and is mandatory in templates for this hook.
- **input_values** contains a parameter called **deploy_identifier**, which stores the **DeployIdentifier** from the parent template. This parameter provides a timestamp to the resource for each deployment update. This ensures the resource reapplies on subsequent overcloud updates.

Next, create an environment file (**/home/stack/templates/pre_config.yaml**) that registers your heat template as the **OS::TripleO::NodeExtraConfig** resource type.

```
resource_registry:
```

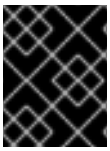
```
  OS::TripleO::NodeExtraConfig: /home/stack/templates/nameserver.yaml

parameter_defaults:
  nameserver_ip: 192.168.1.1
```

To apply the configuration, add the environment file to the stack when creating or updating the Overcloud. For example:

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

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



IMPORTANT

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

6.15. CUSTOMIZING OVERCLOUD POST-CONFIGURATION

A situation might occur where you have completed the creation of your Overcloud but want to add additional configuration, either on initial creation or on a subsequent update of the Overcloud. In this case, you use the **`OS::TripleO::NodeExtraConfigPost`** resource to apply configuration using the standard **`OS::Heat::SoftwareConfig`** types. This applies additional configuration after the main Overcloud configuration completes.

In this example, you first create a basic heat template (**`/home/stack/templates/nameserver.yaml`**) that runs a script to append each node's **`resolv.conf`** with a variable nameserver.

```
heat_template_version: 2014-10-16

description: >
  Extra hostname configuration

parameters:
  servers:
    type: json
  nameserver_ip:
    type: string
  DeployIdentifier:
    type: string

resources:
  ExtraConfig:
    type: OS::Heat::SoftwareConfig
    properties:
      group: script
      config:
        str_replace:
          template: |
            #!/bin/sh
            echo "nameserver _NAMESERVER_IP_" >> /etc/resolv.conf
```

```
          params:
            _NAMESERVER_IP_: {get_param: nameserver_ip}

  ExtraDeployments:
    type: OS::Heat::SoftwareDeployments
    properties:
      config: {get_resource: ExtraConfig}
      servers:  {get_param: servers}
      actions: ['CREATE','UPDATE']
      input_values:
        deploy_identifier: {get_param: DeployIdentifier}
```

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

ExtraConfig

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

ExtraDeployments

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

- The **servers** parameter is provided by the parent template and is mandatory in templates for this hook.
- **input_values** contains a parameter called **deploy_identifier**, which stores the **DeployIdentifier** from the parent template. This parameter provides a timestamp to the resource for each deployment update. This ensures the resource reapplies on subsequent overcloud updates.

Next, create an environment file (**/home/stack/templates/post_config.yaml**) that registers your heat template as the **OS::TripleO::NodeExtraConfigPost:** resource type.

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

parameter_defaults:
  nameserver_ip: 192.168.1.1
```

To apply the configuration, add the environment file to the stack when creating or updating the Overcloud. For example:

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

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

IMPORTANT

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

6.16. CUSTOMIZING PUPPET CONFIGURATION DATA

The Heat template collection contains a set of parameters to pass extra configuration to certain node types. These parameters save the configuration as hieradata for the node's Puppet configuration. These parameters are:

ExtraConfig

Configuration to add to all nodes.

controllerExtraConfig

Configuration to add to all Controller nodes.

NovaComputeExtraConfig

Configuration to add to all Compute nodes.

BlockStorageExtraConfig

Configuration to add to all Block Storage nodes.

ObjectStorageExtraConfig

Configuration to add to all Object Storage nodes

CephStorageExtraConfig

Configuration to add to all Ceph Storage nodes

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

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

Include this environment file when running **openstack overcloud deploy**.

IMPORTANT

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

6.17. APPLYING CUSTOM PUPPET CONFIGURATION

In certain circumstances, you might need to install and configure some additional components to your Overcloud nodes. You can achieve this with a custom Puppet manifest that applies to nodes on after the main configuration completes. As a basic example, you might intend to install **motd** to each node. The process for accomplishing is to first create a Heat template (**/home/stack/templates/custom_puppet_config.yaml**) that launches Puppet configuration.

```
heat_template_version: 2014-10-16

description: >
  Run Puppet extra configuration to set new MOTD

parameters:
  servers:
    type: json
```

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

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

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

Next, create an environment file (**/home/stack/templates/puppet_post_config.yaml**) that registers your heat template as the **OS::TripleO::NodeExtraConfigPost:** resource type.

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

And finally include this environment file when creating or updating the Overcloud stack:

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

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

6.18. USING CUSTOMIZED CORE HEAT TEMPLATES

When creating the Overcloud, the director uses a core set of heat templates. You can copy the standard heat templates into a local directory and use these templates for creating your Overcloud.

Copy the heat template collection in **/usr/share/openstack-tripleo-heat-templates** to the **stack** user's templates directory:

```
$ cp -r /usr/share/openstack-tripleo-heat-templates ~/templates/my-
overcloud
```

This creates a clone of the Overcloud Heat templates. When running **openstack overcloud deploy**, we use the **--templates** option to specify your local template directory. This occurs later in this guide (see Chapter 7, *Creating the Overcloud*).

NOTE

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

IMPORTANT

Red Hat provides updates to the heat template collection over subsequent releases. Using a modified template collection can lead to a divergence between your custom copy and the original copy in **`/usr/share/openstack-tripleo-heat-templates`**. Red Hat recommends using the methods from the following section instead of modifying the heat template collection:

- Section 6.14, “Customizing Overcloud Pre-Configuration”
- Section 6.15, “Customizing Overcloud Post-Configuration”
- Section 6.16, “Customizing Puppet Configuration Data”
- Section 6.17, “Applying Custom Puppet Configuration”

If creating a copy of the heat template collection, you should track changes to the templates using a version control system such as **`git`**.

CHAPTER 7. CREATING THE OVERCLOUD

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. This chapter 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.

7.1. SETTING OVERCLOUD PARAMETERS

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

Table 7.1. Deployment Parameters

Parameter	Description	Example
--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/`**	~/templates/my-overcloud
--stack STACK	The name of the stack to create or update	overcloud
-t [TIMEOUT], --timeout [TIMEOUT]	Deployment timeout in minutes	240
--control-scale [CONTROL_SCALE]	The number of Controller nodes to scale out	3
--compute-scale [COMPUTE_SCALE]	The number of Compute nodes to scale out	3
--ceph-storage-scale [CEPH_STORAGE_SCALE]	The number of Ceph Storage nodes to scale out	3
--block-storage-scale [BLOCK_STORAGE_SCALE]	The number of Cinder nodes to scale out	3

--swift-storage-scale [SWIFT_STORAGE_SCALE]	The number of Swift nodes to scale out	3
--control-flavor [CONTROL_FLAVOR]	The flavor to use for Controller nodes	control
--compute-flavor [COMPUTE_FLAVOR]	The flavor to use for Compute nodes	compute
--ceph-storage-flavor [CEPH_STORAGE_FLAVOR]	The flavor to use for Ceph Storage nodes	ceph-storage
--block-storage-flavor [BLOCK_STORAGE_FLAVOR]	The flavor to use for Cinder nodes	cinder-storage
--swift-storage-flavor [SWIFT_STORAGE_FLAVOR]	The flavor to use for Swift storage nodes	swift-storage
--neutron-flat-networks [NEUTRON_FLAT_NETWORKS]	(DEPRECATED) Defines the flat networks to configure in neutron plugins. Defaults to "datacentre" to permit external network creation	datacentre
--neutron-physical-bridge [NEUTRON_PHYSICAL_BRIDGE]	(DEPRECATED) An Open vSwitch bridge to create on each hypervisor. This defaults to "br-ex". Typically, this should not need to be changed	br-ex
--neutron-bridge-mappings [NEUTRON_BRIDGE_MAPPINGS]	(DEPRECATED) 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	datacentre:br-ex
--neutron-public-interface [NEUTRON_PUBLIC_INTERFACE]	(DEPRECATED) Defines the interface to bridge onto br-ex for network nodes	nic1, eth0
--neutron-network-type [NEUTRON_NETWORK_TYPE]	(DEPRECATED) The tenant network type for Neutron	gre or vxlan
--neutron-tunnel-types [NEUTRON_TUNNEL_TYPES]	(DEPRECATED) The tunnel types for the Neutron tenant network. To specify multiple values, use a comma separated string	*vxlan gre,vxlan*

--neutron-tunnel-id-ranges [NEUTRON_TUNNEL_ID_RANGES]	(DEPRECATED) Ranges of GRE tunnel IDs to make available for tenant network allocation	1:1000
--neutron-vni-ranges [NEUTRON_VNI_RANGES]	(DEPRECATED) Ranges of VXLAN VNI IDs to make available for tenant network allocation	1:1000
--neutron-disable-tunneling	(DEPRECATED) Disables tunneling in case you aim to use a VLAN segmented network or flat network with Neutron	
--neutron-network-vlan-ranges [NEUTRON_NETWORK_VLAN_RANGES]	(DEPRECATED) The Neutron ML2 and Open vSwitch VLAN mapping range to support. Defaults to permitting any VLAN on the *datacentre* physical network	datacentre:1:1000
--neutron-mechanism-drivers [NEUTRON_MECHANISM_DRIVERS]	(DEPRECATED) The mechanism drivers for the neutron tenant network. Defaults to "openvswitch". To specify multiple values, use a comma-separated string	*openvswitch,l2population*
--libvirt-type [LIBVIRT_TYPE]	Virtualization type to use for hypervisors	kvm,qemu
--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.	pool.ntp.org
--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.	ocuser
-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.	-e ~/templates/my-config.yaml
--environment-directory	The directory containing environment files to include in deployment. The command processes these environment files in numerical, then alphabetical order.	--environment-directory ~/templates
--validation-errors-fatal	The Overcloud creation process performs a set of pre-deployment checks. This option exits if any 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.	
--force-postconfig	Force the Overcloud post-deployment configuration.	--force-postconfig
--answers-file ANSWERS_FILE	Path to a YAML file with arguments and parameters.	--answers-file ~/answers.yaml
--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	
--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	

NOTE

Run the following command for a full list of options:

```
$ openstack help overcloud deploy
```

7.2. 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:

- Any network isolation files, including the initialization file (**environments/network-isolation.yaml**) from the heat template collection and then your custom NIC configuration file. See Section 6.2, “Isolating Networks” for more information on network isolation.
- 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.

Likewise, 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
10-network-isolation.yaml
20-network-environment.yaml
30-storage-environment.yaml
40-rhel-registration.yaml
```

The director requires these environment files for re-deployment and post-deployment functions in Chapter 8, *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.

IMPORTANT

Save the original deployment command for later use and modification. For example, save your deployment command in a script file called **deploy-overcloud.sh**:

```
#!/bin/bash
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 \
  -t 150 \
  --control-scale 3 \
  --compute-scale 3 \
  --ceph-storage-scale 3 \
  --swift-storage-scale 0 \
  --block-storage-scale 0 \
  --compute-flavor compute \
  --control-flavor control \
  --ceph-storage-flavor ceph-storage \
  --swift-storage-flavor swift-storage \
  --block-storage-flavor block-storage \
  --ntp-server pool.ntp.org \
  --libvirt-type qemu
```

This retains the Overcloud deployment command's parameters and environment files for future use, such as Overcloud modifications and scaling. You can then edit and rerun this script to suit future customizations to the Overcloud.

7.3. OVERCLOUD CREATION EXAMPLE

The following command is an example of how to start the Overcloud creation with custom environment files included:

```
$ 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 \
```

This command contains the following additional options:

- **--templates** - Creates the Overcloud using the Heat template collection in **/usr/share/openstack-tripleo-heat-templates**.
- **-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 the network environment file from Section 6.2.2, “Creating a Network Environment File”.
- **`-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.
- **`--control-scale 3`** - Scale the Controller nodes to three.
- **`--compute-scale 3`** - Scale the Compute nodes to three.
- **`--ceph-storage-scale 3`** - Scale the Ceph Storage nodes to three.
- **`--control-flavor control`** - Use the a specific flavor for the Controller nodes.
- **`--compute-flavor compute`** - Use the a specific flavor for the Compute nodes.
- **`--ceph-storage-flavor ceph-storage`** - Use the a specific flavor for the Ceph Storage nodes.
- **`--ntp-server pool.ntp.org`** - Use an NTP server for time synchronization. This is useful for keeping the Controller node cluster in synchronization.

7.4. 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                # Initializes the stack user to use the
CLI commands
$ heat stack-list --show-nested
```

The **`heat stack-list --show-nested`** command shows the current stage of the Overcloud creation.

7.5. 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
```

7.6. COMPLETING THE OVERCLOUD CREATION

This concludes the creation of the Overcloud. For post-creation functions, see Chapter 8, *Performing Tasks after Overcloud Creation*.

CHAPTER 8. PERFORMING TASKS AFTER OVERCLOUD CREATION

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

8.1. 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
$ neutron net-create default
$ neutron subnet-create --name default --gateway 172.20.1.1 default
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**:

```
$ neutron net-list
+----------------------+-------------+------------------------------
---------------------+
| id                   | name        | subnets
|
+----------------------+-------------+------------------------------
---------------------+
| 95fadaa1-5dda-4777... | default     | 7e060813-35c5-462c-a56a-
1c6f8f4f332f 172.20.0.0/16 |
+----------------------+-------------+------------------------------
---------------------+
```

8.2. CREATING THE OVERCLOUD EXTERNAL NETWORK

You previously configured the node interfaces to use the External network in Section 6.2, “Isolating Networks”. However, you still need to create this 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
$ neutron net-create public --router:external --provider:network_type flat
--provider:physical_network datacentre
$ neutron subnet-create --name public --enable_dhcp=False --allocation-
pool=start=10.1.1.51,end=10.1.1.250 --gateway=10.1.1.1 public 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 8.5, “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
$ neutron net-create public --router:external --provider:network_type vlan
--provider:physical_network datacentre --provider:segmentation_id 104
$ neutron subnet-create --name public --enable_dhcp=False --allocation-
pool=start=10.1.1.51,end=10.1.1.250 --gateway=10.1.1.1 public 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**:

```
$ neutron net-list
+----------------------+-------------+------------------------------
--------------------+
| id                   | name        | subnets
|
+----------------------+-------------+------------------------------
--------------------+
| d474fe1f-222d-4e32... | public      | 01c5f621-1e0f-4b9d-9c30-
7dc59592a52f 10.1.1.0/24  |
+----------------------+-------------+------------------------------
--------------------+
```

8.3. 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:

```
$ neutron net-create ext-net --router:external --provider:physical_network
floating --provider:network_type vlan --provider:segmentation_id 105
$ neutron subnet-create --name ext-subnet --enable_dhcp=False --
allocation-pool start=10.1.2.51,end=10.1.2.250 --gateway 10.1.2.1 ext-net
10.1.2.0/24
```

8.4. 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:

```
$ neutron net-create --provider:physical_network datacentre --
provider:network_type vlan --provider:segmentation_id 201 --shared
provider_network
```

This command creates a shared network. It is also possible to specify a tenant instead of specifying --shared. 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:

```
$ neutron subnet-create --name provider-subnet --enable_dhcp=True --
allocation-pool start=10.9.101.50,end=10.9.101.100 --gateway 10.9.101.254
provider_network 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:

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

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:

```
$ neutron router-interface-add external subnet1
```

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

8.5. VALIDATING THE OVERCLOUD

The Overcloud uses Tempest to conduct a series of integration tests. This procedure shows how to validate your Overcloud using Tempest. If running this test from the Undercloud, ensure 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 Tempest, 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:

```
$ keystone role-create --name heat_stack_owner
```

Install the Tempest toolset:

```
$ sudo yum install openstack-tempest
```

Set up a **tempest** directory in your **stack** user's home directory and copy a local version of the Tempest suite:

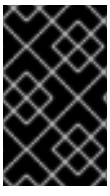
```
$ mkdir ~/tempest
$ cd ~/tempest
$ /usr/share/openstack-tempest-10.0.0/tools/configure-tempest-directory
```

This creates a local version of the Tempest tool set.

After the Overcloud creation process completed, the director created a file named **~/tempest-deployer-input.conf**. This file provides a set of Tempest configuration options relevant to your Overcloud. Run the following command to use this file to configure Tempest:

```
$ tools/config_tempest.py --deployer-input ~/tempest-deployer-input.conf --
--debug --create identity.uri $OS_AUTH_URL identity.admin_password
$OS_PASSWORD --network-id d474fe1f-222d-4e32-9242-cd1fefe9c14b
```

The **$OS_AUTH_URL** and **$OS_PASSWORD** environment variables use values set from the **overcloudrc** file sourced previously. The **--network-id** is the UUID of the external network created in Section 8.2, "Creating the Overcloud External Network".

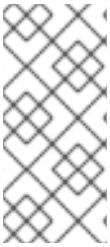


IMPORTANT

The configuration script downloads the Cirros image for the Tempest tests. Make sure the director has access to the Internet or uses a proxy with access to the Internet. Set the **http_proxy** environment variable to use a proxy for command line operations.

Run the full suite of Tempest tests with the following command:

```
$ tools/run-tests.sh
```



NOTE

The full Tempest test suite might take hours. Alternatively, run part of the tests using the **`'.*smoke'`** option.

```
$ tools/run-tests.sh '.*smoke'
```

Each test runs against the Overcloud, and the subsequent output displays each test and its result. You can see more information about each test in the **`tempest.log`** file generated in the same directory. For example, the output might show the following failed test:

```
{2}
tempest.api.compute.servers.test_servers.ServersTestJSON.test_create_speci
fy_keypair [18.305114s] ... FAILED
```

This corresponds to a log entry that contains more information. Search the log for the last two parts of the test namespace separated with a colon. In this example, search for **`ServersTestJSON:test_create_specify_keypair`** in the log:

```
$ grep "ServersTestJSON:test_create_specify_keypair" tempest.log -A 4
2016-03-17 14:49:31.123 10999 INFO tempest_lib.common.rest_client [req-
a7a29a52-0a52-4232-9b57-c4f953280e2c ] Request
(ServersTestJSON:test_create_specify_keypair): 500 POST
http://192.168.201.69:8774/v2/2f8bef15b284456ba58d7b149935cbc8/os-keypairs
4.331s
2016-03-17 14:49:31.123 10999 DEBUG tempest_lib.common.rest_client [req-
a7a29a52-0a52-4232-9b57-c4f953280e2c ] Request - Headers: {'Content-Type':
'application/json', 'Accept': 'application/json', 'X-Auth-Token':
'<omitted>'}
        Body: {"keypair": {"name": "tempest-key-722237471"}}
    Response - Headers: {'status': '500', 'content-length': '128', 'x-
compute-request-id': 'req-a7a29a52-0a52-4232-9b57-c4f953280e2c',
'connection': 'close', 'date': 'Thu, 17 Mar 2016 04:49:31 GMT', 'content-
type': 'application/json; charset=UTF-8'}
        Body: {"computeFault": {"message": "The server has either erred or
is incapable of performing the requested operation.", "code": 500}}
_log_request_full /usr/lib/python2.7/site-
packages/tempest_lib/common/rest_client.py:414
```

NOTE

The **`-A 4`** option shows the next four lines, which are usually the request header and body, then the response header and body.

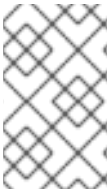
After completing the validation, remove any temporary connections to the Overcloud's Internal API. In this example, use the following commands to remove the previously created VLAN on the Undercloud:

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

8.6. 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 8.1. 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
```

8.7. 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 Chapter 7, *Creating the Overcloud*, you would rerun the following command:

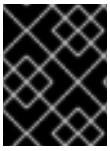
```
$ openstack overcloud deploy --templates -e /usr/share/openstack-tripleo-
heat-templates/environments/network-isolation.yaml -e ~/templates/network-
environment.yaml -e ~/templates/storage-environment.yaml --control-scale 3
--compute-scale 3 --ceph-storage-scale 3 --control-flavor control --
compute-flavor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

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
```

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.

8.8. 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.

```
$ nova image-create instance_name image_name
$ glance image-download image_name --file exported_vm.qcow2
```

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

```
$ glance image-create --name imported_image --file exported_vm.qcow2 --
disk-format qcow2 --container-format bare
$ nova boot --poll --key-name default --flavor m1.demo --image
imported_image --nic net-id=net_id imported
```

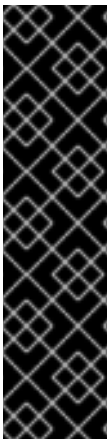
IMPORTANT

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.

8.9. MIGRATING VMS FROM AN OVERCLOUD COMPUTE NODE

In some situations, you might perform maintenance on an Overcloud Compute node. To prevent downtime, migrate the VMs on the Compute node to another Compute node in the Overcloud using the following procedures.

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 automatically.



IMPORTANT

The latest update of Red Hat OpenStack Platform 9 includes patches required for live migration capabilities. The director's core template collection did not include this functionality in the initial release but is now included in the **`openstack-tripleo-heat-templates-2.0.0-57.el7ost`** package and later versions.

Update your environment to use the Heat templates from the **`openstack-tripleo-heat-templates-2.0.0-57.el7ost`** package or later versions.

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

To migrate an instance, source the **`overcloudrc`**, and obtain a list of the current nova services:

```
$ source ~/stack/overcloudrc
$ nova service-list
```

Disable the **`nova-compute`** service on the node you intend to migrate.

```
$ nova service-disable [hostname] nova-compute
```

This prevents new instances from being scheduled on it.

Begin the process of migrating instances off the node. The following command migrates a single instance:

```
$ openstack server migrate [server-name]
```

Run this command for each instance you need to migrate from the disabled Compute node.

Retrieve the current status of the migration process with the following command:

```
$ nova migration-list
```

When migration of each instance completes, its state in nova will change to **VERIFY_RESIZE**. This gives you an opportunity to confirm that the migration completed successfully, or to roll it back. To confirm the migration, use the command:

```
$ nova resize-confirm [server-name]
```

Run this command for each instance with a **VERIFY_RESIZE** status.

This migrates all instances from a host. You can now perform maintenance on the host without any instance downtime. To return the host to an enabled state, run the following command:

```
$ nova service-enable [hostname] nova-compute
```

8.10. 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.

8.11. REMOVING THE OVERCLOUD

The whole Overcloud can be removed when desired.

Delete any existing Overcloud:

```
$ heat stack-delete overcloud
```

Confirm the deletion of the Overcloud:

```
$ heat stack-list
```

Deletion takes a few minutes.

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

CHAPTER 9. SCALING THE OVERCLOUD

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.

WARNING

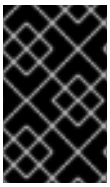
With High Availaibility for Compute instances (or Instance HA, as described in High Availability for Compute Instances), upgrades or scale-up operations are not possible. Any attempts to do so will fail.

If you have Instance HA enabled, disable it before performing an upgrade or scale-up. To do so, perform a *rollback* as described in Rollback.

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:

```
$ ironic node-set-provision-state [NODE UUID] manage
$ openstack baremetal introspection start [NODE UUID]
$ ironic node-set-provision-state [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:

```
$ ironic node-update [NODE UUID] add
properties/capabilities='profile:compute,boot_option:local'
```

Set the boot images to use during the deployment. Find the UUIDs for the **bm-deploy-kernel** and **bm-deploy-ramdisk** images:

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

Set these UUIDs for the new node's **deploy_kernel** and **deploy_ramdisk** settings:

```
$ ironic node-update [NODE UUID] add driver_info/deploy_kernel='09b40e3d-
0382-4925-a356-3a4b4f36b514'
$ ironic node-update [NODE UUID] add driver_info/deploy_ramdisk='765a46af-
4417-4592-91e5-a300ead3faf6'
```

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 Section 8.9, "Migrating VMs from an Overcloud Compute Node" for more details.

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

```
$ source ~/stack/overcloudrc
$ nova service-list
$ nova service-disable [hostname] nova-compute
$ source ~/stack/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:

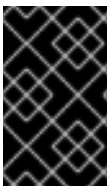
```
$ heat stack-list
```

Identify the UUIDs of the nodes to delete:

```
$ nova list
```

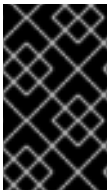
Run the following command to delete the nodes from the stack and update the plan accordingly:

```
$ 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 ~/stack/overcloudrc
$ nova service-list
$ nova service-delete [service-id]
$ source ~/stack/stackrc
```

And remove the node's Open vSwitch agent:

```
$ source ~/stack/overcloudrc
$ neutron agent-list
$ neutron agent-delete [openvswitch-agent-id]
$ source ~/stack/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 Section 8.9, "Migrating VMs from an Overcloud Compute Node" 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
$ heat stack-list --show-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 accomodate 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
$ nova 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. 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 ~]$ nova 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 **ironic node-list**

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

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

-

```
[stack@director ~]$ ironic node-update 75b25e9a-948d-424a-9b3b-
f0ef70a6eacf add
properties/capabilities='profile:control,boot_option:local'
```

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

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

IMPORTANT

If replacing the node with index 0, edit the heat templates and change the bootstrap node index and node validation index before starting replacement. Create a copy of the director's Heat template collection (see Section 6.18, "Using Customized Core Heat Templates") and run the following command on the **overcloud.yaml** file:

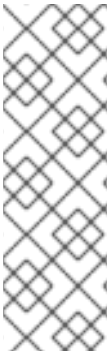
```
$ sudo sed -i "s/resource\.0/resource.1/g" ~/templates/my-
overcloud/overcloud.yaml
```

This changes the node index for the following resources:

```
ControllerBootstrapNodeConfig:
  type: OS::TripleO::BootstrapNode::SoftwareConfig
    properties:
      bootstrap_nodeid: {get_attr: [Controller,
resource.0.hostname]}
      bootstrap_nodeid_ip: {get_attr: [Controller,
resource.0.ip_address]}
```

And:

```
AllNodesValidationConfig:
  type: OS::TripleO::AllNodes::Validation
  properties:
    PingTestIps:
      list_join:
      - ' '
      - - {get_attr: [Controller,
resource.0.external_ip_address]}
        - {get_attr: [Controller,
resource.0.internal_api_ip_address]}
        - {get_attr: [Controller,
resource.0.storage_ip_address]}
        - {get_attr: [Controller,
resource.0.storage_mgmt_ip_address]}
        - {get_attr: [Controller,
resource.0.tenant_ip_address]}
```



NOTE

You can speed up the replacement process by reducing the number for tries for settle in Corosync. Include the following hieradata in the **ExtraConfig** parameter in an environment file:

```
parameter_defaults:
  ExtraConfig:
    pacemaker::corosync::settle_tries: 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 ~]$ heat stack-list --show-nested
```

9.4.3. Manual Intervention

During the **ControllerNodesPostDeployment** stage, the Overcloud stack update halts with an **UPDATE_FAILED** error at **ControllerLoadBalancerDeployment_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 ~]$ nova 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. Enable the keystone service on the new node. Copy the **/etc/keystone** directory from a remaining node to the director host:

```
[heat-admin@overcloud-controller-0 ~]$ sudo -i
[root@overcloud-controller-0 ~]$ scp -r /etc/keystone
stack@192.168.0.1:~/.
```

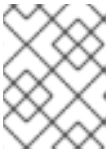
Log in to the new Controller node. Remove the **/etc/keystone** directory from the new Controller node and copy the **keystone** files from the director host:

```
[heat-admin@overcloud-controller-3 ~]$ sudo -i
[root@overcloud-controller-3 ~]$ rm -rf /etc/keystone
[root@overcloud-controller-3 ~]$ scp -r stack@192.168.0.1:~/keystone
```

```
/etc/.
[root@overcloud-controller-3 ~]$ chown -R keystone: /etc/keystone
[root@overcloud-controller-3 ~]$ chown root
/etc/keystone/logging.conf /etc/keystone/default_catalog.templates
```

Edit **/etc/keystone/keystone.conf** and set the **admin_bind_host** and **public_bind_host** parameters to new Controller node's IP addresses. To find these IP addresses, use the **ip addr** command and look for the IP address within the following networks:

- **admin_bind_host** - Provisioning network
- **public_bind_host** - Internal API network



NOTE

These networks might differ if you deployed the Overcloud using a custom **ServiceNetMap** parameter.

For example, if the Provisioning network uses the **192.168.0.0/24** subnet and the Internal API uses the **172.17.0.0/24** subnet, use the following commands to find the node's IP addresses on those networks:

```
[root@overcloud-controller-3 ~]$ ip addr | grep "192\.168\.0\..*/24"
[root@overcloud-controller-3 ~]$ ip addr | grep "172\.17\.0\..*/24"
```

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. Wait until the **httpd** service starts on all nodes.

```
[heat-admin@overcloud-controller-3 ~]$ sudo pcs status | grep httpd
-A1
Clone Set: httpd-clone [httpd]
Started: [ 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
httpd --node overcloud-controller-3
```

18. 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.

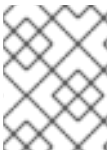
9.4.4. Finalizing Overcloud Services

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.

Exit to the director

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

9.4.5. 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 sure 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
```

9.4.6. 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
```

9.4.7. 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

To replace nodes on the Object Storage cluster, you need to:

- Update the Overcloud with the new Object Storage nodes and prevent Director from creating the ring files.
- Manually add/remove the nodes to the cluster using **`swift-ring-builder`**.

The following procedure describes how to replace nodes while maintaining the integrity of the cluster. In this example, we have a two node Object Storage cluster. The aim is to add an additional node, then replace the faulty node.

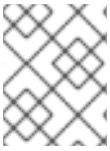
First, create an environment file called **`~/templates/swift-ring-prevent.yaml`** with the following content:

```
parameter_defaults:
  SwiftRingBuild: false
  RingBuild: false
  ObjectStorageCount: 3
```

The **`SwiftRingBuild`** and **`RingBuild`** parameters define whether the Overcloud automatically builds the ring files for Object Storage and Controller nodes respectively. The **`ObjectStorageCount`** defines how many Object Storage nodes in our environment. In this situation, we scale from 2 to 3 nodes.

Include the **`swift-ring-prevent.yaml`** file with the rest of your Overcloud’s environment files as part of the **`openstack overcloud deploy`**:

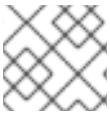
```
$ openstack overcloud deploy --templates [ENVIRONMENT_FILES] -e swift-
ring-prevent.yaml
```



NOTE

Add this file 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. However, the node's storage directory has not been created and ring files for the node's object store are unbuilt. This means you must create the storage directory and build the ring files manually.

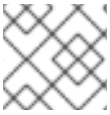


NOTE

Use the following procedure to also build ring files on Controller nodes.

Login to the new node and create the storage directory:

```
$ sudo mkdir -p /srv/node/d1
$ sudo chown -R swift:swift /srv/node/d1
```



NOTE

You can also mount an external storage device at this directory.

Copy the existing ring files to the node. Log into a Controller node as the **heat-admin** user and then change to the superuser. For example, given a Controller node with an IP address of 192.168.201.24.

```
$ ssh heat-admin@192.168.201.24
$ sudo -i
```

Copy the **/etc/swift/*.builder** files from the Controller node to the new Object Storage node's **/etc/swift/** directory. If necessary, transfer the files to the director host:

```
[root@overcloud-controller-0 ~]# scp /etc/swift/*.builder
stack@192.1.2.1:~/.
```

Then transfer the files to the new node:

```
[stack@director ~]$ scp ~/*.builder heat-admin@192.1.2.24:~/.
```

Log into the new Object Storage node as the **heat-admin** user and then change to the superuser. For example, given a Object Storage node with an IP address of 192.168.201.29.

```
$ ssh heat-admin@192.168.201.29
$ sudo -i
```

Copy the files to the **/etc/swift** directory:

```
# cp /home/heat-admin/*.builder /etc/swift/.
```

Add the new Object Storage node to the account, container, and object rings. Run the following commands for the new node:

```
# swift-ring-builder /etc/swift/account.builder add zX-IP:6002/d1 weight
# swift-ring-builder /etc/swift/container.builder add zX-IP:6001/d1 weight
# swift-ring-builder /etc/swift/object.builder add zX-IP:6000/d1 weight
```

Replace the following values in these commands:

zX

Replace X with the corresponding integer of a specified zone (for example, z1 for Zone 1).

IP

The IP that the account, container, and object services use to listen. This should match the IP address of each storage node; specifically, the value of **bind_ip** in the **DEFAULT** sections of **/etc/swift/object-server.conf**, **/etc/swift/account-server.conf**, and **/etc/swift/container-server.conf**.

weight

Describes relative weight of the device in comparison to other devices. This is usually **100**.

NOTE

Check the existing values of the current nodes in the ring file using the **swift-ring-builder** on the rings files alone:

```
# swift-ring-builder /etc/swift/account.builder
```

Remove the node you aim to replace from the account, container, and object rings. Run the following commands for each node:

```
# swift-ring-builder /etc/swift/account.builder remove IP
# swift-ring-builder /etc/swift/container.builder remove IP
# swift-ring-builder /etc/swift/object.builder remove IP
```

Replace **IP** with the IP address of the node.

Redistribute the partitions across all the nodes:

```
# swift-ring-builder /etc/swift/account.builder rebalance
# swift-ring-builder /etc/swift/container.builder rebalance
# swift-ring-builder /etc/swift/object.builder rebalance
```

Change the ownership of all **/etc/swift/** contents to the **root** user and **swift** group:

```
# chown -R root:swift /etc/swift
```

Restart the **openstack-swift-proxy** service:

```
# systemctl restart openstack-swift-proxy.service
```

At this point, the ring files (*.ring.gz and *.builder) should be updated on the new node:

```
/etc/swift/account.builder
/etc/swift/account.ring.gz
```

```
/etc/swift/container.builder
/etc/swift/container.ring.gz
/etc/swift/object.builder
/etc/swift/object.ring.gz
```

Copy these files to **/etc/swift/** on the Controller nodes and the existing Object Storage nodes (except for the node to remove). If necessary, transfer the files to the director host:

```
[root@overcloud-objectstorage-2 swift]# scp *.builder stack@192.1.2.1:~/
[root@overcloud-objectstorage-2 swift]# scp *.ring.gz stack@192.1.2.1:~/
```

Then copy the files to the **/etc/swift/** on each node.

On each node, change the ownership of all **/etc/swift/** contents to the **root** user and **swift** group:

```
# chown -R root:swift /etc/swift
```

The new node is added and a part of the ring. Before removing the old node from the ring, check that the new node completes a full data replication pass.

To remove the old node from the ring, reduce the **ObjectStorageCount** to the omit the old ring. In this case, we reduce from 3 to 2:

```
parameter_defaults:
  SwiftRingBuild: false
  RingBuild: false
  ObjectStorageCount: 2
```

Create a new environment file (**remove-object-node.yaml**) to identify and remove the old Object Storage node. In this case, we remove **overcloud-objectstorage-1**:

```
parameter_defaults:
  ObjectStorageRemovalPolicies:
    [{'resource_list': ['1']}]
```

Include both environment files with the deployment command:

```
$ openstack overcloud deploy --templates -e swift-ring-prevent.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 THE OVERCLOUD

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 Ceph Storage nodes
4. Reboot Compute nodes
5. 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.

When the node boots, check the status of all services:

```
$ sudo systemctl list-units "openstack*" "neutron*" "openvswitch*"
```

Verify the existence of your Overcloud and its nodes:

```
$ source ~/stackrc
$ openstack server list
$ ironic 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 reboot it:

```
$ sudo reboot
```

The remaining Controller Nodes in the cluster retain the high availability services during the reboot.

2. Wait until the node boots.

3. 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.

4. Check all **systemd** services on the Controller Node are active:

```
$ sudo systemctl list-units "openstack*" "neutron*" "openvswitch*"
```

5. 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 CEPH STORAGE NODES

To reboot the Ceph Storage nodes, follow this process:

1. Select the first Ceph Storage node to reboot and log into it.

2. Disable Ceph Storage cluster rebalancing temporarily:

```
$ sudo ceph osd set noout
$ sudo ceph osd set norebalance
```

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, enable cluster rebalancing again:

```
$ sudo ceph osd unset noout
$ sudo ceph osd unset norebalance
```

8. Perform a final status check to make sure the cluster reports **HEALTH_OK**:

```
$ sudo ceph status
```

10.4. 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]
```

3. Select a second Compute Node to act as the target host for migrating instances. This host needs enough resources to host the migrated instances. From the undercloud, migrate each instance from the disabled host to the target host.

```
$ nova live-migration [instance-name] [target-hostname]
$ nova migration-list
$ nova resize-confirm [instance-name]
```

4. Repeat this step until you have migrated all instances from the Compute Node.

IMPORTANT

For full instructions on configuring and migrating instances, see Section 8.9, “Migrating VMs from an Overcloud Compute Node”.

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.

10.5. 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 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.

11.3.1. 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`**.

11.3.2. 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

- 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.

11.3.3. 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 11.5, "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 six main steps for the deployment. This includes:

Table 11.1. Controller Node Configuration Steps

Step	Description
ControllerLoadBalancerDeployment_Step1	Initial load balancing software configuration, including Pacemaker, RabbitMQ, Memcached, Redis, and Galera.
ControllerServicesBaseDeployment_Step2	Initial cluster configuration, including Pacemaker configuration, HAProxy, MongoDB, Galera, Ceph Monitor, and database initialization for OpenStack Platform services.

ControllerRingbuilderDeployment_Step3	Initial ring build for OpenStack Object Storage (**swift**).
ControllerOvercloudServicesDeployment_Step4	Configuration of all OpenStack Platform services (**nova**, **neutron**, **cinder**, **sahara**, **ceilometer**, **heat**, **horizon**, **aodh**, **gnocchi**).
ControllerOvercloudServicesDeployment_Step5	Configure service start up settings in Pacemaker, including constraints to determine service start up order and service start up parameters.
ControllerOvercloudServicesDeployment_Step6	Final pass of the Overcloud configuration.

11.4. 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
```

11.5. 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
   ```

11.6. 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.

11.6.1. 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 11.3.3, “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.

11.6.2. 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/`**.

11.6.3. 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 Section 8.9, “Migrating VMs from an Overcloud Compute Node” for more information on node migrations.

11.6.4. Ceph Storage Service Failures

For any issues that occur with Red Hat Ceph Storage clusters, see Part 10. Logging and Debugging in the Red Hat Ceph Storage Configuration Guide. This section provides information on diagnosing logs for all Ceph storage services.

11.7. 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
```

11.8. 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 Undercloud and Overcloud

Information	Undercloud or Overcloud	Log Location
General director services	Undercloud	**/var/log/nova/*** **/var/log/heat/*** **/var/log/ironic/***
Introspection	Undercloud	**/var/log/ironic/*** **/var/log/ironic-inspector/***
Provisioning	Undercloud	**/var/log/ironic/***
Cloud-Init Log	Overcloud	**/var/log/cloud-init.log**
Overcloud Configuration (Summary of Last Puppet Run)	Overcloud	**/var/lib/puppet/state/last_run_summary.yaml**
Overcloud Configuration (Report from Last Puppet Run)	Overcloud	**/var/lib/puppet/state/last_run_report.yaml**
Overcloud Configuration (All Puppet Reports)	Overcloud	**/var/lib/puppet/reports/overcloud-*/***

General Overcloud services	Overcloud	**/var/log/ceilometer/*** **/var/log/ceph/*** **/var/log/cinder/*** **/var/log/glance/*** **/var/log/heat/*** **/var/log/horizon/*** **/var/log/httpd/*** **/var/log/keystone/*** **/var/log/libvirt/*** **/var/log/neutron/*** **/var/log/nova/*** **/var/log/openvswitch/*** **/var/log/rabbitmq/*** **/var/log/redis/*** **/var/log/swift/***
High availability log	Overcloud	**/var/log/pacemaker.log**

APPENDIX A. SSL/TLS CERTIFICATE CONFIGURATION

As an optional part of the processes outlined in Section 4.6, “Configuring the Director” or Section 6.10, “Enabling SSL/TLS on the Overcloud”, you can set the use SSL/TLS for communication on either the Undercloud or Overcloud. However, if using a SSL certificate with your own certificate authority, the certificate requires the configuration steps in the following section.

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:

```
$ sudo 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 the IP address, or fully qualified domain name if using one, of the Public API:

- For the Undercloud, use the **undercloud_public_vip** parameter in **undercloud.conf**. If using a fully qualified domain name for this IP address, use the domain name instead.
- For the Overcloud, use the IP address for the Public API, which is the first address for the **ExternalAllocationPools** parameter in your network isolation environment file. If using a fully qualified domain name for this IP address, use the domain name instead.

Edit the **alt_names** section to include the following entries:

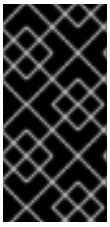
- **IP** - A list of IP addresses for clients to access the director over SSL.
- **DNS** - A list of domain names for clients to access the director over SSL. Also include the Public API IP address as a DNS entry at the end of the **alt_names** section.

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.



IMPORTANT

The **openssl req** command asks for several details for the certificate, including the Common Name. Make sure the Common Name is set to the IP address of the Public API for the Undercloud or Overcloud (depending on which certificate set you are creating). The **openssl.cnf** file should use this IP address as a default value.

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 throught 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”.

A.8. USING THE CERTIFICATE WITH THE OVERCLOUD

Include the contents of your certificate files in the **enable-tls.yaml** environment file from Section 6.10, “Enabling SSL/TLS on the Overcloud”. The Overcloud deployment process takes the parameters from **enable-tls.yaml** and automatically integrates them on each node in the Overcloud.

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)

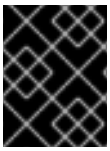
Sensor data retrieval method. Use either **ipmitool** or **scci**. The default is **ipmitool**.

- 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. 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.

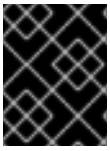
pm_addr

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. 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.

C.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 C.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"
  }
]
```

C.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.

C.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
```

C.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 D. BASE PARAMETERS

The following is a list of base parameters used for configuring the Overcloud. These parameters are defined in the **`parameters`** section of the **`overcloud.yaml`** file in the director's core Heat template collection.

AdminPassword

Type: string
The password for the OpenStack Identity administration account.

AdminToken

Type: string
The OpenStack Identity authentication secret.

AodhPassword

Type: string
The password for the OpenStack Telemetry Alarming services.

BlockStorageCount

Type: number
The number of Block Storage nodes in your Overcloud.

BlockStorageExtraConfig

Type: json
Block Storage specific configuration to inject into the cluster.

BlockStorageHostnameFormat

Type: string
Format for Block Storage node host names.

BlockStorageImage

Type: string
The image to use for provisioning Block Storage nodes.

BlockStorageRemovalPolicies

Type: json
List of resources to be removed from **`BlockStorageResourceGroup`** when doing an update that requires removal of specific resources.

BlockStorageSchedulerHints

Type: json
Optional scheduler hints to pass to OpenStack Compute.

CeilometerBackend

Type: string
The OpenStack Telemetry backend type. Select either **`mongodb`** or **`mysql`**.

CeilometerComputeAgent

Type: string
Indicates whether the Compute agent is present and expects **`nova-compute`** to be configured accordingly.

CeilometerMeterDispatcher

Type: string
The OpenStack Telemetry (**`ceilometer`**) service includes a new component for a time series data storage (**`gnocchi`**). It is possible in Red Hat OpenStack Platform to switch the default Ceilometer dispatcher to use this new component instead of the standard database. You accomplish this with the **`CeilometerMeterDispatcher`**, which you set to either:

- **`database`** - Use the standard database for the Ceilometer dispatcher. This is the default option.
- **`gnocchi`** - Use the new time series database for Ceilometer dispatcher. See **Example 3** in Configuring Base Parameters for more information.

CeilometerMeteringSecret

Type: string
Secret shared by the OpenStack Telemetry services.

CeilometerPassword

Type: string
The password for the OpenStack Telemetry service account.

CephAdminKey

Type: string
The Ceph admin client key. Can be created with **`ceph-authtool --gen-print-key`**.

CephClientKey

Type: string
The Ceph client key. Can be created with **`ceph-authtool --gen-print-key`**. Currently only used for external Ceph deployments to create the OpenStack user keyring.

CephClusterFSID

Type: string
The Ceph cluster FSID. Must be a UUID.

CephExternalMonHost

Type: string
List of externally managed Ceph Monitors host IPs. Only used for external Ceph deployments.

CephMonKey

Type: string
The Ceph Monitors key. Can be created with **`ceph-authtool --gen-print-key`**.

CephStorageCount

Type: number
The number of Ceph Storage nodes in your Overcloud.

CephStorageExtraConfig

Type: json
Ceph Storage specific configuration to inject into the cluster.

CephStorageHostnameFormat

Type: string
Format for Ceph Storage node host names.

CephStorageImage

Type: string
The image to use for provisioning Ceph Storage nodes.

CephStorageRemovalPolicies

Type: json
List of resources to be removed from **`CephStorageResourceGroup`** when doing an update that requires removal of specific resources.

CephStorageSchedulerHints

Type: json
Optional scheduler hints to pass to OpenStack Compute.

CinderEnableIscsiBackend

Type: boolean
Whether to enable or not the iSCSI backend for Block Storage.

CinderEnableNfsBackend

Type: boolean
Whether to enable or not the NFS backend for Block Storage.

CinderEnableRbdBackend

Type: boolean
Whether to enable or not the Ceph Storage backend for Block Storage.

CinderISCSIHelper

Type: string
The iSCSI helper to use with Block Storage.

CinderLVMLoopDeviceSize

Type: number
The size of the loopback file used by the Block Storage LVM driver.

CinderNfsMountOptions

Type: string
Mount options for NFS mounts used by Block Storage NFS backend. Effective when **`CinderEnableNfsBackend`** is true.

CinderNfsServers

Type: comma delimited list

NFS servers used by Block Storage NFS backend. Effective when **`CinderEnableNfsBackend`** is true.

CinderPassword

Type: string
The password for the Block Storage service account.

CloudDomain

Type: string
The DNS domain used for the hosts. This should match the **`dhcp_domain`** configured in the Undercloud's networking. Defaults to **`localdomain`**.

CloudName

Type: string
The DNS name of this cloud. For example: **`ci-overcloud.tripleo.org`**.

ComputeCount

Type: number
The number of Compute nodes in your Overcloud.

ComputeHostnameFormat

Type: string
Format for Compute node host names.

ComputeRemovalPolicies

Type: json
List of resources to be removed from **`ComputeResourceGroup`** when doing an update that requires removal of specific resources.

ControlFixedIPs

Type: json
A list of fixed IP addresses for Controller nodes.

ControlVirtualInterface

Type: string
Interface where virtual IPs are assigned.

ControllerCount

Type: number
The number of Controller nodes in your Overcloud.

ControllerEnableCephStorage

Type: boolean
Whether to deploy Ceph Storage (OSD) on the Controller.

ControllerEnableSwiftStorage

Type: boolean
Whether to enable Object Storage on the Controller.

controllerExtraConfig

Type: json
Controller specific configuration to inject into the cluster.

ControllerHostnameFormat

Type: string
Format for Controller node host names.

controllerImage

Type: string
The image to use for provisioning Block Storage nodes.

ControllerRemovalPolicies

Type: json
List of resources to be removed from **`ControllerResourceGroup`** when doing an update that requires removal of specific resources.

ControllerSchedulerHints

Type: json
Optional scheduler hints to pass to OpenStack Compute.

CorosyncIPv6

Type: boolean
Enable IPv6 in Corosync.

Debug

Type: string
Set to **`true`** to enable debugging on all services.

DeployIdentifier

Type: string
Set to a unique value to re-run any deployment tasks that perform configuration on a **`stack-update`** of the Overcloud.

EnableFencing

Type: boolean
Defines whether to enable fencing in Pacemaker or not.

EnableGalera

Type: boolean
Defines whether to use Galera instead of regular MariaDB.

ExtraConfig

Type: json
Additional configuration to inject into the cluster. Any role-specific **`ExtraConfig`**, such as **`controllerExtraConfig`**, takes precedence over the standard **`ExtraConfig`**.

FencingConfig

Type: json
Pacemaker fencing configuration. The JSON should have the following structure:

```
{
  "devices": [
    {
      "agent": "AGENT_NAME",
      "host_mac": "HOST_MAC_ADDRESS",
      "params": {"PARAM_NAME": "PARAM_VALUE"}
    }
  ]
}
```

For instance:

```
{
  "devices": [
    {
      "agent": "fence_xvm",
      "host_mac": "52:54:00:aa:bb:cc",
      "params": {
        "multicast_address": "225.0.0.12",
        "port": "baremetal_0",
        "manage_fw": true,
        "manage_key_file": true,
        "key_file": "/etc/fence_xvm.key",
        "key_file_password": "abcdef"
      }
    }
  ]
}
```

GlanceBackend

Type: string
The short name of the OpenStack Image backend to use. Should be one of **`swift`**, **`rbd`**, or **`file`**.

GlanceLogFile

Type: string
The path of the file to use for logging messages from OpenStack Image.

GlanceNotifierStrategy

Type: string
Strategy to use for OpenStack Image notification queue. Defaults to **noop**.

GlancePassword

Type: string
The password for the OpenStack Image service account, used by the OpenStack Image services.

GnocchiBackend

Type: string
The short name of the Gnocchi backend to use. Should be one of **`swift`**, **`rbd`**, or **`file`**.

GnocchiIndexerBackend

Type: string
The short name of the Gnocchi indexer backend to use.

GnocchiPassword

Type: string
The password for the Gnocchi service account.

HAProxySyslogAddress

Type: string
Syslog address where HAProxy will send its log.

HeatPassword

Type: string
The password for the Heat service account.

HeatStackDomainAdminPassword

Type: string
Password for **`heat_stack_domain_admin`** user.

HorizonAllowedHosts

Type: comma delimited list
A list of IPs/Hostnames allowed to connect to the OpenStack Dashboard.

HypervisorNeutronPhysicalBridge

Type: string
The OVS bridge to create on each hypervisor. This defaults to **`br-ex`**, which is the same as the control plane nodes, as we have a uniform configuration of the Open vSwitch agent. Typically, you should not need to change this.

HypervisorNeutronPublicInterface

Type: string
Defines what interface to add to the **`HypervisorNeutronPhysicalBridge`**.

ImageUpdatePolicy

Type: string
What policy to use when reconstructing instances. **`REBUILD`** for rebuilds and **`REBUILD_PRESERVE_EPHEMERAL`** to preserve **`/mnt`**.

InstanceNameTemplate

Type: string
Template string to be used to generate instance names.

InternalApiVirtualFixedIPs

Type: json
Control the IP allocation for the InternalApiVirtualInterface port. For example:

```
[{'ip_address':'1.2.3.4'}]
```

KeyName

Type: string
Name of an existing OpenStack Compute key pair to enable SSH access to the instances.

KeystoneCACertificate

Type: string
OpenStack Identity self-signed certificate authority certificate.

KeystoneNotificationDriver

Type: comma delimited list
Comma-separated list of notification drivers used by OpenStack Identity.

KeystoneNotificationFormat

Type: string
The OpenStack Identity notification format.

KeystoneSSLCertificate

Type: string
OpenStack Identity certificate for verifying token validity.

KeystoneSSLCertificateKey

Type: string
OpenStack Identity key for signing tokens.

KeystoneSigningCertificate

Type: string
OpenStack Identity certificate for verifying token validity.

KeystoneSigningKey

Type: string
OpenStack Identity key for signing tokens.

ManageFirewall

Type: boolean
Defines whether to manage firewall rules.

MemcachedIPv6

Type: boolean
Enable IPv6 features in Memcached.

MongoDbIPv6

Type: boolean
Enable IPv6 if MongoDB VIP is IPv6.

MongoDbNoJournal

Type: boolean
Defines if MongoDb journaling is disabled.

MysqlInnodbBufferPoolSize

Type: number
Specifies the size of the buffer pool in megabytes. Setting to zero should be interpreted as "no value" and defers to the lower level default.

MysqlMaxConnections

Type: number
Configures MySQL **`max_connections`** setting.

NeutronAgentExtensions

Type: comma delimited list
Comma-separated list of extensions enabled for the OpenStack Networking agents.

NeutronAgentMode

Type: string
Agent mode for the **`neutron-l3-agent`** on the Controller hosts.

NeutronAllowL3AgentFailover

Type: string
Allow automatic L3 Agent failover.

NeutronBridgeMappings

Type: comma delimited list
The OVS logical-to-physical bridge mappings to use. Defaults to mapping the external bridge on hosts (**`br-ex`**) to a physical name (*datacentre*), which can be used to create provider networks (and we use this for the default floating network). If changing this, either use different post-install network scripts or make sure to keep *datacentre* as a mapping network name.

NeutronComputeAgentMode

Type: string
Agent mode for the **`neutron-l3-agent`** on the Compute nodes.

NeutronControlPlaneID

Type: string
Neutron ID or name for **`ctlplane`** network.

NeutronCorePlugin

Type: string
The core plugin for OpenStack Networking. The value should be the entry point to be loaded from neutron.core_plugins name space.

NeutronDVR

Type: string
Whether to configure OpenStack Networking Distributed Virtual Routers

NeutronDhcpAgentsPerNetwork

Type: number
The number of OpenStack Networking dhcp agents to schedule per network

NeutronDnsmasqOptions

Type: string
Dnsmasq options for **`neutron-dhcp-agent`**. The default value here forces MTU to be set to the value of **`NeutronTenantMtu`**, which should be set to account for tunnel overhead.

NeutronEnableIsolatedMetadata

Type: string
If true, DHCP provide metadata route to VM.

NeutronEnableL2Pop

Type: string
Enable/disable the L2 population feature in the OpenStack Networking agents.

NeutronEnableTunnelling

Type: string
Defines whether to enable tunneling in OpenStack Networking.

NeutronExternalNetworkBridge

Type: string
Name of bridge used for external network traffic.

NeutronFlatNetworks

Type: comma delimited list
Defines a list of flat networks to configure in OpenStack Networking plugins. Defaults to *datacentre* to permit external network creation.

NeutronL3HA

Type: string
If you need to disable Layer 3 High Availability (L3HA) for OpenStack Networking, set this to **`false`** in an environment file.

NeutronMechanismDrivers

Type: comma delimited list
The mechanism drivers for the OpenStack Networking tenant network.

NeutronMetadataProxySharedSecret

Type: string
Shared secret to prevent spoofing

NeutronNetworkType

Type: comma delimited list
The tenant network type for OpenStack Networking.

NeutronNetworkVLANRanges

Type: comma delimited list
The OpenStack Networking ML2 and OpenVSwitch vlan mapping range to support. See the OpenStack Networking documentation for permitted values. Defaults to permitting any VLAN on the *datacentre* physical network (See **`NeutronBridgeMappings`**).

NeutronPassword

Type: string
The password for the OpenStack Networking service account, used by OpenStack Networking agents.

NeutronPluginExtensions

Type: comma delimited list
Comma-separated list of extensions enabled for the OpenStack Networking plugin.

NeutronPublicInterface

Type: string
What interface to bridge on to **`br-ex`** for network nodes.

NeutronPublicInterfaceDefaultRoute

Type: string
A custom default route for the **`NeutronPublicInterface`**.

NeutronPublicInterfaceIP

Type: string
A custom IP address to put onto the **`NeutronPublicInterface`**.

NeutronPublicInterfaceRawDevice

Type: string
If this is set, the public interface will be a VLAN which uses this device as the raw device.

NeutronPublicInterfaceTag

Type: string
VLAN tag for creating a public VLAN. The tag will be used to create an access port on the exterior bridge for each control plane node and gives that port the IP address returned from the OpenStack Networking public network.

NeutronServicePlugins

Type: comma delimited list
Comma-separated list of service plugin entry points to be loaded from the neutron.service_plugins name space.

NeutronTenantMtu

Type: string
The default MTU for tenant networks. For VXLAN/GRE tunneling, this should be at least 50 bytes smaller than the MTU on the physical network. This value will be used to set the MTU on the virtual Ethernet device. This value will be used to construct the **`NeutronDnsmasqOptions`**, since that will determine the MTU that is assigned to the VM host through DHCP.

NeutronTunnelIdRanges

Type: comma delimited list
Comma-separated list of **`<tun_min>:<tun_max>`** tuples enumerating ranges of GRE tunnel IDs that are available for tenant network allocation.

NeutronTunnelTypes

Type: comma delimited list
The tunnel types for the OpenStack Networking tenant network.

NeutronTypeDrivers

Type: comma delimited list
Comma-separated list of network type driver entry points to be loaded.

NeutronVniRanges

Type: comma delimited list
Comma-separated list of **<vni_min>:<vni_max>** tuples enumerating ranges of VXLAN VNI IDs that are available for tenant network allocation

NovaComputeDriver

Type: string
The OpenStack Compute driver to use for managing instances. Defaults to the **libvirt.LibvirtDriver** driver.

NovaComputeExtraConfig

Type: json
Compute node specific configuration to inject into the cluster.

NovaComputeLibvirtType

Type: string
Defines the Libvirt type to use. Defaults to **kvm**.

NovaComputeLibvirtVifDriver

Type: string
Libvirt VIF driver configuration for the network.

NovaComputeSchedulerHints

Type: json
Optional scheduler hints to pass to OpenStack Compute.

NovaEnableRbdBackend

Type: boolean
Whether to enable or not the Ceph backend for Nova.

NovaIPv6

Type: boolean
Enable IPv6 features in Nova.

NovaImage

Type: string
The image to use for provisioning Compute nodes.

NovaOVSBridge

Type: string
Name of integration bridge used by Open vSwitch.

NovaPassword

Type: string
The password for the OpenStack Compute service account.

NovaSecurityGroupAPI

Type: string
The full class name of the security API class.

NtpServer

Type: comma delimited list
Comma-separated list of NTP servers.

ObjectStorageCount

Type: number
The number of Object Storage nodes in your Overcloud.

ObjectStorageExtraConfig

Type: json
ObjectStorage specific configuration to inject into the cluster.

ObjectStorageHostnameFormat

Type: string
Format for Object Storage node host names.

ObjectStorageRemovalPolicies

Type: json
List of resources to be removed from **`ObjectStorageResourceGroup`** when doing an update that requires removal of specific resources.

ObjectStorageSchedulerHints

Type: json
Optional scheduler hints to pass to OpenStack Compute.

OvercloudBlockStorageFlavor

Type: string
Flavor for Block Storage nodes to request when deploying.

OvercloudCephStorageFlavor

Type: string
Flavor for Ceph Storage nodes to request when deploying.

OvercloudComputeFlavor

Type: string
Flavor for Compute nodes to request when deploying.

OvercloudControlFlavor

Type: string
Flavor for Controller nodes to request when deploying.

OvercloudSwiftStorageFlavor

Type: string
Flavor for Object Storage nodes to request when deploying.

PublicVirtualFixedIPs

Type: json
Control the IP allocation for the **`PublicVirtualInterface`** port. For example:

```
[{'ip_address':'1.2.3.4'}]
```

PublicVirtualInterface

Type: string
Specifies the interface where the public-facing virtual IP will be assigned. This should be **`int_public`** when a VLAN is being used.

PurgeFirewallRules

Type: boolean
Defines whether to purge firewall rules before setting up new ones.

RabbitClientPort

Type: number
Set RabbitMQ subscriber port.

RabbitClientUseSSL

Type: string
RabbitMQ client subscriber parameter to specify an SSL connection to the RabbitMQ host.

RabbitCookieSalt

Type: string
Salt for the RabbitMQ cookie. Change this to force the randomly generated RabbitMQ cookie to change.

RabbitFDLimit

Type: string
Configures RabbitMQ file descriptor limit.

RabbitIPv6

Type: boolean
Enable IPv6 in RabbitMQ.

RabbitPassword

Type: string
The password for RabbitMQ.

RabbitUserName

Type: string
The username for RabbitMQ

RedisPassword

Type: string
The password for Redis

SaharaPassword

Type: string
The password for the OpenStack Clustering service account.

ServerMetadata

Type: json
Extra properties or metadata passed to OpenStack Compute for the created nodes in the Overcloud.

ServiceNetMap

Type: json
Mapping of service names to network names. Typically set in the **`parameter_defaults`** of the resource registry.

SnmpdReadonlyUserName

Type: string
The user name for SNMPd with read-only rights running on all Overcloud nodes.

SnmpdReadonlyUserPassword

Type: string
The user password for SNMPd with read-only rights running on all Overcloud nodes.

StorageMgmtVirtualFixedIPs

Type: json
Control the IP allocation for the StorageMgmgVirtualInterface port. For example:

```
[{'ip_address':'1.2.3.4'}]
```

StorageVirtualFixedIPs

Type: json
Control the IP allocation for the **`StorageVirtualInterface`** port. For example:

```
[{'ip_address':'1.2.3.4'}]
```

SwiftHashSuffix

Type: string
A random string to be used as a salt when hashing to determine mappings in the ring.

SwiftMinPartHours

Type: number
The minimum time (in hours) before a partition in a ring can be moved following a rebalance.

SwiftMountCheck

Type: boolean

Value of **mount_check** in Object Storage account/container/object-server.conf.

SwiftPartPower

Type: number
Partition power to use when building Object Storage rings.

SwiftPassword

Type: string
The password for the Object Storage service account, used by the Object Storage proxy services.

SwiftReplicas

Type: number
How many replicas to use in the Object Storage rings.

SwiftStorageImage

Type: string
The image to use for provisioning Object Storage nodes.

TimeZone

Type: string
Sets the time zone of your Overcloud deployment. If you leave the **TimeZone** parameter blank, the Overcloud will default to **UTC** time. Director recognizes the standard timezone names defined in the timezone database */usr/share/zoneinfo/*. For example, if you wanted to set your time zone to **Japan**, you would examine the contents of */usr/share/zoneinfo* to locate a suitable entry:

```
$ ls /usr/share/zoneinfo/
Africa      Asia       Canada   Cuba   EST      GB       GMT-0
HST      iso3166.tab  Kwajalein  MST      NZ-CHAT   posix       right
Turkey     UTC       Zulu
America     Atlantic   CET      EET    EST5EDT  GB-Eire  GMT+0
Iceland  Israel       Libya      MST7MDT  Pacific   posixrules  ROC
UCT        WET
Antarctica  Australia  Chile    Egypt  Etc      GMT      Greenwich
Indian   Jamaica      MET        Navajo   Poland    PRC         ROK
Universal  W-SU
Arctic      Brazil     CST6CDT  Eire   Europe   GMT0     Hongkong
Iran     Japan        Mexico     NZ       Portugal  PST8PDT
Singapore  US         zone.tab
```

The output listed above includes time zone files, and directories containing additional time zone files. For example, **Japan** is an individual time zone file in this result, but **Africa** is a directory containing additional time zone files:

```
$ ls /usr/share/zoneinfo/Africa/
Abidjan      Algiers  Bamako  Bissau       Bujumbura   Ceuta
Dar_es_Salaam  El_Aaiun  Harare        Kampala   Kinshasa    Lome
Lusaka  Maseru     Monrovia  Niamey       Porto-Novo  Tripoli
Accra        Asmara   Bangui  Blantyre     Cairo       Conakry
Djibouti       Freetown  Johannesburg  Khartoum  Lagos       Luanda
Malabo  Mbabane    Nairobi   Nouakchott   Sao_Tome    Tunis
```

```
Addis_Ababa  Asmera   Banjul  Brazzaville  Casablanca  Dakar    Douala
Gaborone  Juba          Kigali    Libreville  Lubumbashi  Maputo
Mogadishu  Ndjamena  Ouagadougou  Timbuktu    Windhoek
```

UpdateIdentifier

Type: string
Setting to a previously unused value during **`stack-update`** triggers a package update on all nodes.

APPENDIX E. NETWORK INTERFACE PARAMETERS

The following tables define the Heat template parameters for network interface types.

E.1. INTERFACE OPTIONS

Option	Default	Description
name		Name of the Interface
use_dhcp	False	Use DHCP to get an IP address
use_dhcpv6	False	Use DHCP to get a v6 IP address
addresses		A sequence of IP addresses assigned to the interface
routes		A sequence of routes assigned to the interface
mtu	1500	The maximum transmission unit (MTU) of the connection
primary	False	Defines the interface as the primary interface
defroute	True	Use this interface as the default route
persist_mapping	False	Write the device alias configuration instead of the system names
dhclient_args	None	Arguments to pass to the DHCP client
dns_servers	None	List of DNS servers to use for the interface

E.2. VLAN OPTIONS

Option	Default	Description
vlan_id		The VLAN ID

device		The VLAN's parent device to attach the VLAN. For example, use this parameter to attach the VLAN to a bonded interface device.
use_dhcp	False	Use DHCP to get an IP address
use_dhcpv6	False	Use DHCP to get a v6 IP address
addresses		A sequence of IP addresses assigned to the VLAN
routes		A sequence of routes assigned to the VLAN
mtu	1500	The maximum transmission unit (MTU) of the connection
primary	False	Defines the VLAN as the primary interface
defroute	True	Use this interface as the default route
persist_mapping	False	Write the device alias configuration instead of the system names
dhclient_args	None	Arguments to pass to the DHCP client
dns_servers	None	List of DNS servers to use for the VLAN

E.3. OVS BOND OPTIONS

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

routes		A sequence of routes assigned to the bond
mtu	1500	The maximum transmission unit (MTU) of the connection
primary	False	Defines the interface as the primary interface
members		A sequence of interface objects to use in the bond
ovs_options		A set of options to pass to OVS when creating the bond
ovs_extra		A set of options to to set as the OVS_EXTRA parameter in the bond's network configuration file
defroute	True	Use this interface as the default route
persist_mapping	False	Write the device alias configuration instead of the system names
dhclient_args	None	Arguments to pass to the DHCP client
dns_servers	None	List of DNS servers to use for the bond

E.4. OVS BRIDGE OPTIONS

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

mtu	1500	The maximum transmission unit (MTU) of the connection
members		A sequence of interface, VLAN, and bond objects to use in the bridge
ovs_options		A set of options to pass to OVS when creating the bridge
ovs_extra		A set of options to to set as the OVS_EXTRA parameter in the bridge's network configuration file
defroute	True	Use this interface as the default route
persist_mapping	False	Write the device alias configuration instead of the system names
dhclient_args	None	Arguments to pass to the DHCP client
dns_servers	None	List of DNS servers to use for the bridge

E.5. LINUX BOND OPTIONS

Option	Default	Description
name		Name of the bond
use_dhcp	False	Use DHCP to get an IP address
use_dhcpv6	False	Use DHCP to get a v6 IP address
addresses		A sequence of IP addresses assigned to the bond
routes		A sequence of routes assigned to the bond
mtu	1500	The maximum transmission unit (MTU) of the connection

primary	False	Defines the interface as the primary interface
members		A sequence of interface objects to use in the bond
bonding_options		A set of options when creating the bond. For more information on Linux bonding options, see 4.5.1. Bonding Module Directives in the Red Hat Enterprise Linux 7 Networking Guide.
defroute	True	Use this interface as the default route
persist_mapping	False	Write the device alias configuration instead of the system names
dhclient_args	None	Arguments to pass to the DHCP client
dns_servers	None	List of DNS servers to use for the bond

E.6. LINUX BRIDGE OPTIONS

Option	Default	Description
name		Name of the bridge
use_dhcp	False	Use DHCP to get an IP address
use_dhcpv6	False	Use DHCP to get a v6 IP address
addresses		A sequence of IP addresses assigned to the bridge
routes		A sequence of routes assigned to the bridge
mtu	1500	The maximum transmission unit (MTU) of the connection
members		A sequence of interface, VLAN, and bond objects to use in the bridge

defroute	True	Use this interface as the default route
persist_mapping	False	Write the device alias configuration instead of the system names
dhclient_args	None	Arguments to pass to the DHCP client
dns_servers	None	List of DNS servers to use for the bridge

APPENDIX F. NETWORK INTERFACE TEMPLATE EXAMPLES

This appendix provides a few example Heat templates to demonstrate network interface configuration.

F.1. CONFIGURING INTERFACES

Individual interfaces might require modification. The example below shows modifications required to use the second NIC to connect to an infrastructure network with DHCP addresses, and to use the third and fourth NICs for the bond:

```
network_config:
  # Add a DHCP infrastructure network to nic2
  -
    type: interface
    name: nic2
    use_dhcp: true
  -
    type: ovs_bridge
    name: br-bond
    members:
      -
        type: ovs_bond
        name: bond1
        ovs_options: {get_param: BondInterfaceOvsOptions}
        members:
          # Modify bond NICs to use nic3 and nic4
          -
            type: interface
            name: nic3
            primary: true
          -
            type: interface
            name: nic4
```

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

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

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

The numbered NIC scheme only takes into account the interfaces that are live, for example, if they have a cable attached to the switch. If you have some hosts with four interfaces and some with six interfaces, you should use **nic1** to **nic4** and only plug four cables on each host.

F.2. CONFIGURING ROUTES AND DEFAULT ROUTES

There are two ways a host has default routes set. If the interface is using DHCP and the DHCP server offers a gateway address, the system uses a default route for that gateway. Otherwise, you can set a default route on an interface with a static IP.

Although the Linux kernel supports multiple default gateways, it only uses the one with the lowest metric. If there are multiple DHCP interfaces, this can result in an unpredictable default gateway. In this case, it is recommended to set **defroute=no** for interfaces other than the one using the default route.

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

```
# No default route on this DHCP interface
- type: interface
  name: nic2
  use_dhcp: true
  defroute: false
# Instead use this DHCP interface as the default route
- type: interface
  name: nic3
  use_dhcp: true
```

NOTE

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

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

```
    - type: vlan
      device: bond1
      vlan_id: {get_param: InternalApiNetworkVlanID}
      addresses:
      - ip_netmask: {get_param: InternalApiIpSubnet}
      routes:
      - ip_netmask: 10.1.2.0/24
        next_hop: 172.17.0.1
```

F.3. USING THE NATIVE VLAN FOR FLOATING IPS

Neutron uses a default empty string for its external bridge mapping. This maps the physical interface to the **br-int** instead of using **br-ex** directly. This model allows multiple Floating IP networks using either VLANs or multiple physical connections.

Use the **NeutronExternalNetworkBridge** parameter in the **parameter_defaults** section of your network isolation environment file:

```
  parameter_defaults:
    # Set to "br-ex" when using floating IPs on the native VLAN
    NeutronExternalNetworkBridge: "''"
```

Using only one Floating IP network on the native VLAN of a bridge means you can optionally set the neutron external bridge. This results in the packets only having to traverse one bridge instead of two, which might result in slightly lower CPU usage when passing traffic over the Floating IP network.

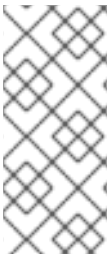
The next section contains changes to the NIC config to put the External network on the native VLAN. If the External network is mapped to **br-ex**, you can use the External network for Floating IPs in addition to the horizon dashboard, and Public APIs.

F.4. USING THE NATIVE VLAN ON A TRUNKED INTERFACE

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

For example, if the External network is on the native VLAN, a bonded configuration looks like this:

```
network_config:
  - type: ovs_bridge
    name: {get_input: bridge_name}
    dns_servers: {get_param: DnsServers}
    addresses:
      - ip_netmask: {get_param: ExternalIpSubnet}
    routes:
      - ip_netmask: 0.0.0.0/0
        next_hop: {get_param: ExternalInterfaceDefaultRoute}
    members:
      - type: ovs_bond
        name: bond1
        ovs_options: {get_param: BondInterfaceOvsOptions}
        members:
          - type: interface
            name: nic3
            primary: true
          - type: interface
            name: nic4
```



NOTE

When moving the address (and possibly route) statements onto the bridge, remove the corresponding VLAN interface from the bridge. Make the changes to all applicable roles. The External network is only on the controllers, so only the controller template requires a change. The Storage network on the other hand is attached to all roles, so if the Storage network is on the default VLAN, all roles require modifications.

F.5. CONFIGURING JUMBO FRAMES

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

The MTU of a VLAN cannot exceed the MTU of the physical interface. Make sure to include the MTU value on the bond and/or interface.

The Storage, Storage Management, Internal API, and Tenant networking all benefit from jumbo frames. In testing, Tenant networking throughput was over 300% greater when using jumbo frames in conjunction with VXLAN tunnels.

NOTE

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

```
- type: ovs_bond
  name: bond1
  mtu: 9000
  ovs_options: {get_param: BondInterfaceOvsOptions}
  members:
    - type: interface
      name: nic3
      mtu: 9000
      primary: true
    - type: interface
      name: nic4
      mtu: 9000

# The external interface should stay at default
- type: vlan
  device: bond1
  vlan_id: {get_param: ExternalNetworkVlanID}
  addresses:
    - ip_netmask: {get_param: ExternalIpSubnet}
  routes:
    - ip_netmask: 0.0.0.0/0
      next_hop: {get_param: ExternalInterfaceDefaultRoute}

# MTU 9000 for Internal API, Storage, and Storage Management
- type: vlan
  device: bond1
  mtu: 9000
  vlan_id: {get_param: InternalApiNetworkVlanID}
  addresses:
  - ip_netmask: {get_param: InternalApiIpSubnet}
```

APPENDIX G. NETWORK ENVIRONMENT OPTIONS

Table G.1. Network Environment Options

Parameter	Description	Example
InternalApiNetCidr	The network and subnet for the Internal API network	172.17.0.0/24
StorageNetCidr	The network and subnet for the Storage network	
StorageMgmtNetCidr	The network and subnet for the Storage Management network	
TenantNetCidr	The network and subnet for the Tenant network	
ExternalNetCidr	The network and subnet for the External network	
InternalApiAllocationPools	The allocation pool for the Internal API network in a tuple format	[\{*start*: *172.17.0.10*, *end*: *172.17.0.200*}]
StorageAllocationPools	The allocation pool for the Storage network in a tuple format	
StorageMgmtAllocationPools	The allocation pool for the Storage Management network in a tuple format	
TenantAllocationPools	The allocation pool for the Tenant network in a tuple format	
ExternalAllocationPools	The allocation pool for the External network in a tuple format	
InternalApiNetworkVlanID	The VLAN ID for the Internal API network	200
StorageNetworkVlanID	The VLAN ID for the Storage network	
StorageMgmtNetworkVlanID	The VLAN ID for the Storage Management network	
TenantNetworkVlanID	The VLAN ID for the Tenant network	

Parameter	Description	Example
ExternalNetworkVlanID	The VLAN ID for the External network	
ExternalInterfaceDefaultRoute	The gateway IP address for the External network	10.1.2.1
ControlPlaneDefaultRoute	Gateway router for the Provisioning network (or Undercloud IP)	ControlPlaneDefaultRoute: 192.0.2.254
ControlPlaneSubnetCidr	CIDR subnet mask length for provisioning network	ControlPlaneSubnetCidr: 24
EC2MetadataIp	The IP address of the EC2 metadata server. Generally the IP of the Undercloud.	EC2MetadataIp: 192.0.2.1
DnsServers	Define the DNS servers for the Overcloud nodes. Include a maximum of two.	DnsServers: ["8.8.8.8","8.8.4.4"]
BondInterfaceOvsOptions	The options for bonding interfaces	BondInterfaceOvsOptions:"bond_mode=balance-slb"
NeutronFlatNetworks	Defines the flat networks to configure in neutron plugins. Defaults to "datacentre" to permit external network creation	NeutronFlatNetworks: "datacentre"
NeutronExternalNetworkBridge	An Open vSwitch bridge to create on each hypervisor. This defaults to "br-ex". Set to **`"br-ex"`** if using floating IPs on native VLAN on bridge **`br-ex`**. Typically, this should not need to be changed.	NeutronExternalNetworkBridge: "br-ex"
NeutronBridgeMappings	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: "datacentre:br-ex"
NeutronPublicInterface	Defines the interface to bridge onto br-ex for network nodes	NeutronPublicInterface: "eth0"

Parameter	Description	Example
NeutronNetworkType	The tenant network type for Neutron	NeutronNetworkType: "vxlan"
NeutronTunnelTypes	The tunnel types for the neutron tenant network. To specify multiple values, use a comma separated string.	NeutronTunnelTypes: *gre,vxlan*
NeutronTunnelIdRanges	Ranges of GRE tunnel IDs to make available for tenant network allocation	NeutronTunnelIdRanges "1:1000"
NeutronVniRanges	Ranges of VXLAN VNI IDs to make available for tenant network allocation	NeutronVniRanges: "1:1000"
NeutronEnableTunnelling	Defines whether to enable or disable tunneling in case you aim to use a VLAN segmented network or flat network with Neutron. Defaults to enabled	
NeutronNetworkVLANRanges	The neutron ML2 and Open vSwitch VLAN mapping range to support. Defaults to permitting any VLAN on the *datacentre* physical network.	NeutronNetworkVLANRanges: "datacentre:1:1000"
NeutronMechanismDrivers	The mechanism drivers for the neutron tenant network. Defaults to "openvswitch". To specify multiple values, use a comma-separated string	NeutronMechanismDrivers: *openvswitch,l2population*

APPENDIX H. OPEN VSWITCH BONDING OPTIONS

The Overcloud provides networking through Open vSwitch (OVS), which provides several options for bonded interfaces. In Section 6.2.2, “Creating a Network Environment File”, you can configure a bonded interface in the network environment file using the following parameter:

```
BondInterfaceOvsOptions:
  "bond_mode=balance-slb"
```

The following table provides some explanation of these options and some alternatives depending on your hardware.

IMPORTANT

Do not use LACP with OVS-based bonds, as this configuration is problematic and unsupported. Instead, consider using *bond_mode=balance-slb* as a replacement for this functionality. In addition, you can still use LACP with Linux bonding in your network interface templates. For example:

```
      - type: linux_bond
        name: bond1
        members:
        - type: interface
          name: nic2
        - type: interface
          name: nic3
        bonding_options: "mode=802.3ad lacp_rate=[fast|slow]
updelay=1000 miimon=100"
```

- **`mode`** - enables LACP.
- **`lacp_rate`** - defines whether LACP packets are sent every 1 second, or every 30 seconds.
- **`updelay`** - defines the minimum amount of time that an interface must be active before it is used for traffic (this helps mitigate port flapping outages).
- **`miimon`** - the interval in milliseconds that is used for monitoring the port state using the driver’s MIIMON functionality.

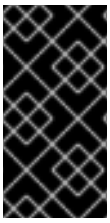
For more information on Linux bonding options, see 4.5.1. Bonding Module Directives in the *Red Hat Enterprise Linux 7 Networking Guide*.

For the technical details behind this requirement, see BZ#1267291.

Table H.1. Bonding Options

bond_mode=balance-slb	Balances flows based on source MAC address and output VLAN, with periodic rebalancing as traffic patterns change. Bonding with **balance-slb** allows a limited form of load balancing without the remote switch's knowledge or cooperation. SLB assigns each source MAC and VLAN pair to a link and transmits all packets from that MAC and VLAN through that link. This mode uses a simple hashing algorithm based on source MAC address and VLAN number, with periodic rebalancing as traffic patterns change. This mode is similar to mode 2 bonds used by the Linux bonding driver. This mode is used when the switch is configured with bonding but is not configured to use LACP (static instead of dynamic bonds).
bond_mode=active-backup	This mode offers active/standby failover where the standby NIC resumes network operations when the active connection fails. Only one MAC address is presented to the physical switch. This mode does not require any special switch support or configuration, and works when the links are connected to separate switches. This mode does not provide load balancing.
lacp=[active|passive|off]	Controls the Link Aggregation Control Protocol (LACP) behavior. Only certain switches support LACP. If your switch does not support LACP, use **bond_mode=balance-slb** or **bond_mode=active-backup**. Do not use LACP with OVS-based bonds, as this configuration is problematic and unsupported. Instead, consider using *bond_mode=balance-slb* as a replacement for this functionality. In addition, you can still use LACP with Linux bonding. For the technical details behind this requirement, see BZ#1267291.
other-config:lacp-fallback-ab=true	Sets the LACP behavior to switch to bond_mode=active-backup as a fallback.
other_config:lacp-time=[fast|slow]	Set the LACP heartbeat to 1 second (fast) or 30 seconds (slow). The default is slow.
other_config:bond-detect-mode=[miimon|carrier]	Set the link detection to use miimon heartbeats (miimon) or monitor carrier (carrier). The default is carrier.
other_config:bond-miimon-interval=100	If using miimon, set the heartbeat interval in milliseconds.

other_config:bond_updelay=1000	Number of milliseconds a link must be up to be activated to prevent flapping.
other_config:bond-rebalance-interval=10000	Milliseconds between rebalancing flows between bond members. Set to zero to disable.



IMPORTANT

If you experience packet drops or performance issues using Linux bonds with Provider networks, consider disabling Large Receive Offload (LRO) on the standby interfaces. Avoid adding a Linux bond to an OVS bond, as port-flapping and loss of connectivity can occur. This is a result of a packet-loop through the standby interface.