Red Hat Enterprise Linux OpenStack Platform 7

Director Installation and Usage

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

Last Updated: 2018-05-21

Red Hat Enterprise Linux OpenStack Platform 7 Director Installation and Usage

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

OpenStack Documentation Team
Red Hat Customer Content Services
rhos-docs@redhat.com

Legal Notice

Abstract

This guide explains how to install Red Hat Enterprise Linux OpenStack Platform 7 in an enterprise environment using the Red Hat Enterprise Linux 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 Enterprise Linux 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 Enterprise Linux OpenStack Platform environment that is both lean and robust.

The Red Hat Enterprise Linux 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.

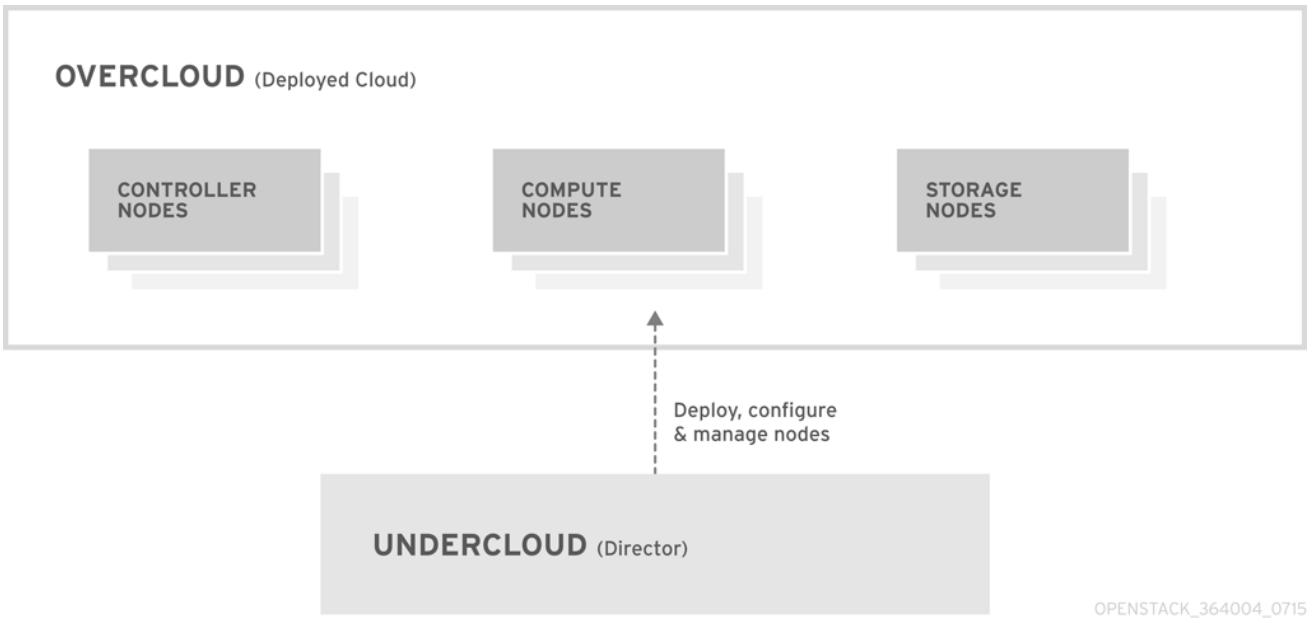


Figure 1.1. Basic Layout of Undercloud and Overcloud

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 Enterprise Linux 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 Enterprise Linux OpenStack Platform director utilizes these Undercloud functions through both a web-based graphical user interface and a terminal-based command line interface.

The Undercloud uses the following components:

- OpenStack Dashboard (Horizon) - The web-based dashboard for the director.
- 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 Server (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) - For monitoring and data collection.
- OpenStack Identity (Keystone) - Authentication for the director's components.
- MariaDB - Database for the director.
- RabbitMQ - Messaging queue for the director's components.

1.2. OVERCLOUD

The Overcloud is the resulting Red Hat Enterprise Linux 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: Horizon, Keystone, Nova API, Neutron Server, Open vSwitch, Glance, Cinder Volume, Cinder API, Swift Storage, Swift Proxy, Heat Engine, Heat API, Ceilometer, MariaDB, RabbitMQ. The Controller also uses Pacemaker and Galera for high availability functions.

- Compute - Nodes used to provide computing resources for the OpenStack environment. Add more Compute nodes to scale your environment over time.

  A default Compute node contains the following components: Nova Compute, Nova KVM, 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 on to 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: Cinder Volume, 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: swift storage, ceilometer agent, Open vSwitch.

1.3. HIGH AVAILABILITY

High availability provides continuous operation to a system or components set through an extended length of time. The Red Hat Enterprise Linux OpenStack Platform director provides high availability to an OpenStack Platform environment through the use of a Controller node cluster. The director installs a set of the same components on each Controller node and manages them as one whole service. Having a cluster provides a fallback in case 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 machines in a cluster.
- HA Proxy - Provides load balancing and proxy services to the cluster.
- Galera - Provides replication of the OpenStack Platform database across the cluster.
- Memcached - Provides database caching.

NOTE

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 with its own peculiar needs consumes block storage resources. Deploying Glance (images), Cinder (volumes) and/or Nova (compute) on a single node becomes impossible to manage in large deployments with thousands of clients or more. Scaling OpenStack externally resolves this challenge.

However, it also makes it a practical requirement to virtualize the storage layer with a solution like Red Hat Ceph Storage so you can scale the Red Hat Enterprise Linux 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 Enterprise Linux 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.

2.1. ENVIRONMENT REQUIREMENTS

Minimum Requirements

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

Recommended Requirements

- 1 host machine for the Red Hat Enterprise Linux OpenStack Platform director
- 3 host machines for Red Hat Enterprise Linux OpenStack Platform Compute nodes
- 3 host machines for Red Hat Enterprise Linux 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.

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. 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 installed as the host operating system.

2.3. NETWORKING REQUIREMENTS

The Undercloud host requires at least two networks:

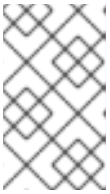
- Provisioning Network - This is a private network the director uses to provision and manage the Overcloud nodes. The Provisioning network provides DHCP and PXE boot functions to help discover bare metal systems for use in the Overcloud. This network must use a native VLAN on a trunked interface so that the director serves PXE boot and DHCP requests. 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. This 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 Enterprise Linux OpenStack Platform network traffic into other networks. Red Hat Enterprise Linux OpenStack Platform supports both physical interfaces and tagged VLANs for network isolation. For more information on network isolation, see Section 4.2, “Planning Networks”.

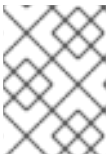
Note the following:

- All machines require at least two NICs. In a typical minimal configuration, use either:
  - One NIC for the Provisioning network and the other NIC for the External network.
  - 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.
- 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, we refer to NICs using a single name across all Overcloud machines. Ideally, you should use the same NIC on each system 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 in the range for scaling the environment.



NOTE

Duplicate IP addresses should be avoided on the Provisioning network. For more information, see Section 12.4, “Avoid IP address conflicts on the Provisioning network”.



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 PXE boot for Provisioning NIC is at the top of the boot order, ahead of hard disks and CD/DVD drives.
- All Overcloud bare metal systems require an Intelligent Platform Management Interface (IPMI) connected to the Provisioning network. The director controls the power management of each node.
- Make a note of the following details for each Overcloud system: the MAC address of the Provisioning NIC, the IP address of the IPMI NIC, IPMI username, and IPMI password. This information is useful to have when setting up the Overcloud nodes.
- 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.

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 minimized. For example:

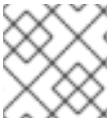
- Use network segmentation to mitigate network movement and isolate sensitive data; a flat network is much less secure.
- Restrict services access and ports to a minimum.
- Ensure proper firewall rules and password usage.
- Ensure that SELinux is enabled.

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.

Intelligent Platform Management Interface (IPMI)

Each Compute node requires IPMI functionality on the server's motherboard.

2.4.2. Controller Node Requirements

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

Processor

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

Memory

A minimum of 6 GB of RAM.

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.

Intelligent Platform Management Interface (IPMI)

Each Controller node requires IPMI functionality on the server's motherboard.

2.4.3. Ceph Storage Node Requirements

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

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.

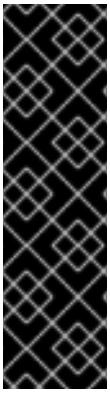
This guide contains the necessary instructions to map your Ceph Storage disks into the director.

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.

Intelligent Platform Management Interface (IPMI)

Each Ceph node requires 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
```

2.5. 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 repositories necessary 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-6.1-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 7.0 director for RHEL 7 (RPMs)	**rhel-7-server-openstack-7.0-director-rpms**	Red Hat OpenStack Platform director repository.
Red Hat Enterprise Linux OpenStack Platform 7.0 for RHEL 7 (RPMs)	**rhel-7-server-openstack-7.0-rpms**	Core Red Hat OpenStack Platform repository.
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.

Name	Repository	Description of Requirement
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.

CHAPTER 3. INSTALLING THE UNDERCLOUD

The first step to creating your Red Hat Enterprise Linux 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.

3.1. 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.

3.2. 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.

3.3. SETTING THE HOSTNAME FOR THE SYSTEM

The director requires a fully qualified domain name for its installation and configuration process. This means you must set for the hostname for 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**, **/etc/hosts** requires an entry like:

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

3.4. 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.

Procedure 3.1. Subscribing to the Required Channels Using Subscription Manager

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

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

2. Find the entitlement pool for the Red Hat Enterprise Linux OpenStack Platform director.

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

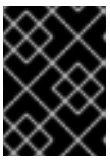
3. Use the pool ID located in the previous step to attach the Red Hat Enterprise Linux OpenStack Platform 7 entitlements:

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

4. Disable all default repositories 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-
   7.0-rpms --enable=rhel-7-server-openstack-7.0-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.

5. 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.

3.5. INSTALLING THE DIRECTOR PACKAGES

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

```
[stack@director ~]$ sudo yum install -y python-rdomanager-oscplugin
```

This installs all packages required for the director installation.

3.6. 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 your are using a different subnet for the Provisioning network i.e. it conflicts with an existing IP address or subnet in your environment.

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 B, *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**. 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 **discovery_interface** parameter.

masquerade_network

Defines the network to 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 to your nodes.

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.

network_gateway

The gateway for the Overcloud instances. This is the discovery 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.

discovery_interface

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

discovery_iprange

A range of IP address that the director's discovery 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**.

discovery_runbench

Runs a set of benchmarks during node discovery. Set to **1** to enable. This option is necessary if you aim to perform benchmark analysis when inspecting the hardware of registered nodes in the Advanced Scenario. See Section 6.2.3, "Automatically Tagging Nodes with Automated Health Check (AHC) Tools" for more details.

undercloud_debug

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

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.

3.7. OBTAINING IMAGES FOR OVERCLOUD NODES

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

- A discovery kernel and ramdisk - Used for bare metal system discovery 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 Red Hat Enterprise Linux OpenStack Platform downloads page on the Red Hat Customer Portal at https://access.redhat.com/downloads/content/191/ver=7/rhel---7/7/x86_64/product-downloads. This location on the Customer Portal contains the images in TAR archives.

Download these image archives to the **images** directory on the **stack** user's home on the directory host (**/home/stack/images/**) and extract the images from the archives:

```
$ cd ~/images
$ for tarfile in *.tar; do tar -xf $tarfile; done
```

Run the following command to 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 discovery images on the director's PXE server.

View a list of the images in the CLI using the following command:

```
$ 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 discovery PXE images (**discovery-ramdisk.***). The director copies these files to **/httpboot**.

```
[stack@host1 ~]$ ls -l /httpboot
total 151636
-rw-r--r--. 1 ironic ironic       269 Sep 19 02:43 boot.ipxe
-rw-r--r--. 1 root   root         252 Sep 10 15:35 discoverd.ipxe
-rwxr-xr-x. 1 root   root     5027584 Sep 10 16:32 discovery.kernel
-rw-r--r--. 1 root   root   150230861 Sep 10 16:32 discovery.ramdisk
drwxr-xr-x. 2 ironic ironic      4096 Sep 19 02:45 pxelinux.cfg
```

NOTE

Ironic manages the **boot.ipxe** file and the **pxelinux.cfg** directory in **/httpboot** during introspection and provisioning. These files might not appear when you view this directory.

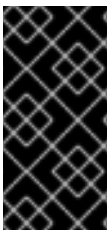
3.8. SETTING A NAMESERVER ON THE UNDERCLOUD'S NEUTRON SUBNET

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 on to separate networks, the Overcloud nodes use the **DnsServer** parameter in your network environment templates. This is covered in the Advanced Overcloud scenario in Section 6.2.6.2, “Creating an Advanced Overcloud Network Environment File”.

3.9. COMPLETING THE UNDERCLOUD CONFIGURATION

This completes the Undercloud configuration. The next few chapter discuss planning and creating your Overcloud.

CHAPTER 4. PLANNING YOUR OVERCLOUD

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

4.1. 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.

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 host that acts as a hypervisor and provides the processing capabilities required for running virtual machines in the environment. A basic Red Hat Enterprise Linux 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.

Cinder-Storage

A host that provides external block storage for OpenStack's Cinder service. This deployment role is optional.

Swift-Storage

A host that provides external object storage for OpenStack's Swift service. This deployment role is optional.

This guide provides multiple scenarios based on the desired environment you want. The following table defines the node types for each scenario.

Table 4.1. Node Deployment Roles for Scenarios

	Controller	Compute	Ceph-Storage	Swift-Storage	Cinder-Storage	Total
Basic Environment	1	1	-	-	-	2
Advanced Environment with Ceph Storage	3	3	3	-	-	9

4.2. 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 Enterprise Linux OpenStack Platform 7 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.

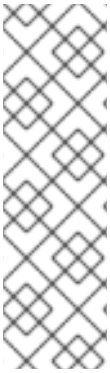
Red Hat Enterprise Linux OpenStack Platform maps different services to separate network traffic types assigned to the various subnets in your environments. These network traffic types include:

Table 4.2. 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 via 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 multiple tenant networks may use the same addresses.	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

Network Type	Description	Used By
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; this traffic connects directly to Ceph.	Controller, Ceph Storage, Cinder Storage, Swift Storage
External	Hosts the OpenStack Dashboard (horizon) for graphical system management, 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, 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 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

In a typical Red Hat Enterprise Linux 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 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 disruption.

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

- Internal API
- Storage
- Storage Management
- Tenant Networks
- External

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`**.

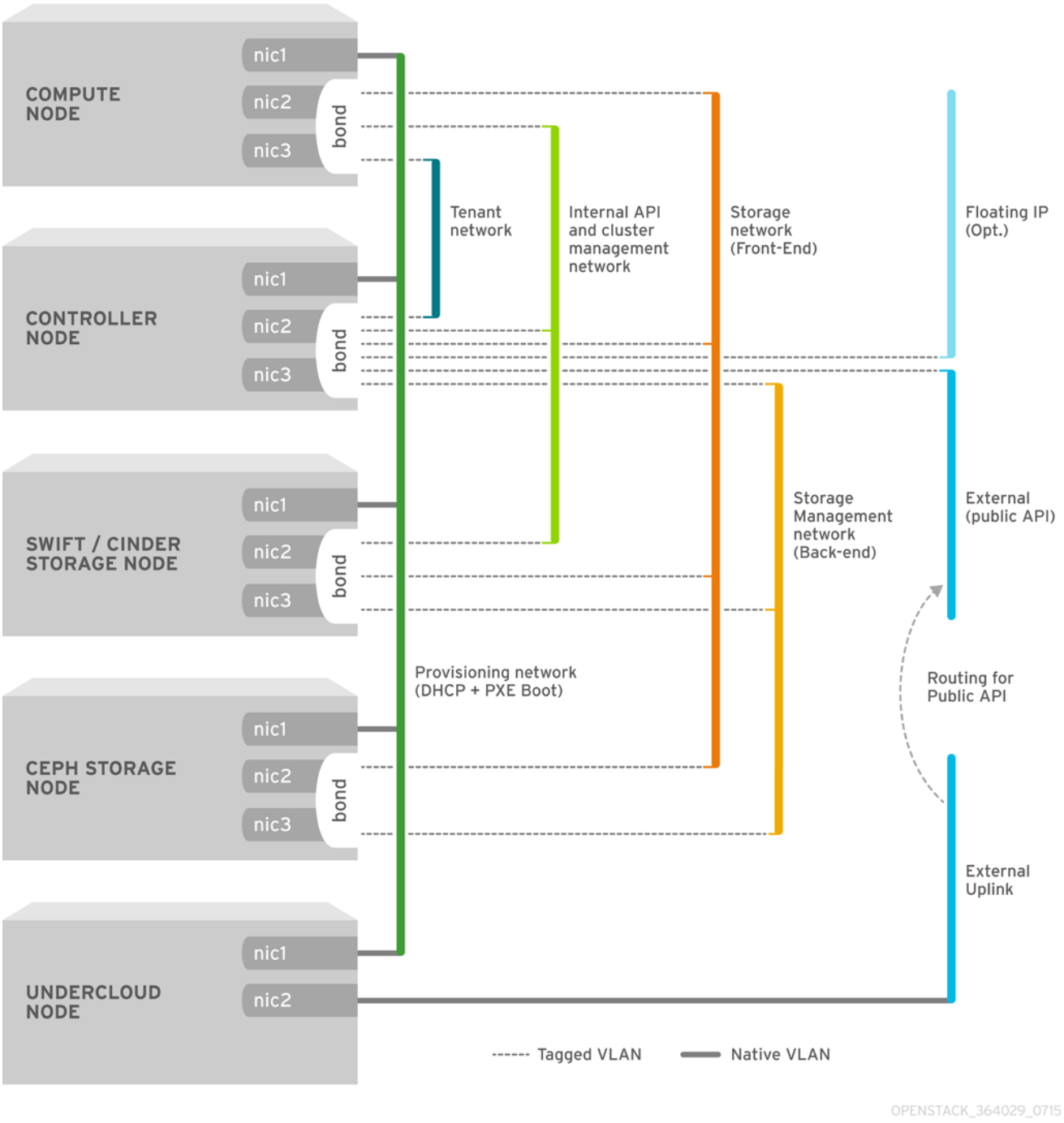


Figure 4.1. Example VLAN Topology using Bonded Interfaces

This guide provides multiple scenarios based on the desired environment you want. The following table defines the network traffic mappings for each scenario:

Table 4.3. Network Mappings

	Mappings	Total Interfaces	Total VLANs
Basic Environment	Network 1 - Provisioning, Internal API, Storage, Storage Management, Tenant Networks Network 2 - External, Floating IP (mapped after Overcloud creation)	2	2

	Mappings	Total Interfaces	Total VLANs
Advanced Environment with Ceph Storage	Network 1 - Provisioning Network 2 - Internal API Network 3 - Tenant Networks Network 4 - Storage Network 5 - Storage Management Network 6 - External, Floating IP (mapped after Overcloud creation)	3 (includes 2 bonded interfaces)	6

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

Cinder Storage Nodes

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

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 5. UNDERSTANDING HEAT TEMPLATES

Some of the scenarios in this guide use custom Heat templates to define certain aspects of the Overcloud, such as network isolation and network interface configuration. This section provides a basic introduction on Heat templates so that you can understand the structure and format of these templates in the context of the Red Hat Enterprise Linux OpenStack Platform director.

5.1. 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 per 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 **`My Cirros Instance`**.

5.2. ENVIRONMENT FILES

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

- **Parameters** - These are common settings you apply to a template's parameters. These are defined in the **`parameters`** section of an environment file.
- **Parameter Defaults** - These parameters modify the default values for parameters in your templates. These are defined in the **`parameter_defaults`** section of an environment file.
- **Resource Registry** - This section defines custom resource names, link 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.

Here is 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
```

This 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.

5.3. DEFAULT DIRECTOR PLANS

The director contains a Heat template collection within its database. This is stored as a *plan*. To view a list of plans in the director:

```
$ openstack management plan list
```

This shows one plan: **`overcloud`**, which is our Overcloud configuration. To view more details in the Overcloud plan:

```
$ openstack management plan show [UUID]
```

You can also download and view the Heat template files for the Overcloud plan. Use the following commands to download the Heat templates from the plan into a directory in the **stack** users **templates** directory.

```
$ mkdir ~/templates/overcloud-plan
$ openstack management plan download [UUID] -O ~/templates/overcloud-plan/
```

This collection contains the main Heat template (**plan.yaml**) and an environment file (**environment.yaml**). The template collection also contains various directories and template files registered as resources in the environment file.

This plan-based template is used to create the Overcloud in the Test Overcloud Scenario.

5.4. DEFAULT DIRECTOR TEMPLATES

The director also contains an advanced 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 three main files to note in this template collection:

- **overcloud-without-mergepy.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.
- **overcloud-resource-registry.yaml** - This is a standard environment file used to create the Overcloud environment. The **overcloud-resource-registry-puppet.yaml** is based on this file. This file is used for a customized configuration of your environment.

The Basic and Advanced Overcloud scenarios use this template collection. Both use the **overcloud-without-mergepy.yaml** template and the **overcloud-resource-registry-puppet.yaml** environment file to configure the Overcloud image for each node. We will also create an environment file to configure network isolation for both the Basic and Advanced Scenarios.

CHAPTER 6. INSTALLING THE OVERCLOUD

Our Undercloud is now installed with the Red Hat Enterprise Linux OpenStack Platform director configured. In this chapter, we use the director to create our Overcloud environment. To help users at various levels, we provide two different installation scenarios to create an Overcloud. Each scenario varies in complexity and topics.

Table 6.1. Scenario Overview

Scenario	Level	Topics
Basic Overcloud	Medium	CLI tool usage, node registration, manual node tagging, basic network isolation, plan-based Overcloud creation
Advanced Overcloud	High	CLI tool usage, node registration, automatic node tagging based on hardware, Ceph Storage setup, advanced network isolation, Overcloud creation, high availability fencing configuration

6.1. BASIC SCENARIO: CREATING A SMALL OVERCLOUD WITH NFS STORAGE

This scenario creates a small enterprise-level OpenStack Platform environment. This scenario consists of two nodes in the Overcloud: one Controller node and one Compute node. Both machines are bare metal systems using IPMI for power management. This scenario focuses on the command line tools to demonstrate the director's ability to create a small production-level Red Hat Enterprise Linux OpenStack Platform environment that can scale Compute nodes in the future.

Workflow

1. Create a node definition template and register blank nodes in the director.
2. Inspect hardware of all nodes.
3. Manually tag nodes into roles.
4. Create flavors and tag them into roles.
5. Create Heat templates to isolate the External network.
6. Create the Overcloud environment using the default Heat template collection and the additional network isolation templates.

Requirements

- The director node created in Chapter 3, *Installing the Undercloud*
- Two bare metal machines. These machines must comply with the requirements set for the Controller and Compute nodes. For these requirements, see:

- Section 2.4.2, “Controller Node Requirements”
- Section 2.4.1, “Compute Node Requirements”

These nodes do not require an operating system because 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 this example, we use 192.0.2.0/24 as the Provisioning subnet with the following IP address assignments:

Table 6.2. Provisioning Network IP Assignments

Node Name	IP Address	MAC Address	IPMI IP Address
Director	192.0.2.1	aa:aa:aa:aa:aa:aa	
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

- One network connection for our External network. All Controller nodes must connect to this network. For this example, we use 10.1.1.0/24 for the External network.
- All other network types use the Provisioning network for OpenStack services
- This scenario also uses an NFS share on a separate server on the Provisioning network. The IP Address for this server is 192.0.2.230.

6.1.1. Registering Nodes for the Basic Overcloud

In this section, we create a node definition template. This file (**`instackenv.json`**) is a JSON format file and contains the hardware and power management details for our two nodes.

This template uses the following attributes:

mac

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

pm_type

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

pm_user, pm_password

The IPMI username and password.

pm_addr

The IP address of the IPMI device.

cpu

The number of CPUs on the node.

memory

The amount of memory in MB.

disk

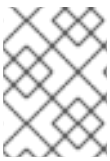
The size of the hard disk in GB.

arch

The system architecture.

For example:

```
{
    "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"
        }
    ]
}
```



NOTE

For more supported power management types and their options, see Appendix C, *Power Management Drivers*.

After creating the template, save the file to the **stack** user's home directory (**/home/stack/instackenv.json**), then import it into the director. Use the following command to accomplish this:

```
$ 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 using the following command:

```
$ openstack baremetal list
```

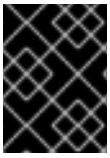
6.1.2. Inspecting the Hardware of Nodes

After registering the nodes, we inspect the hardware attribute of each node. 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-discoverd -u openstack-ironic-
discoverd-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 maintenance mode, perform the introspection, then revert the node out of maintenance mode:

```
$ ironic node-set-maintenance [NODE UUID] true
$ openstack baremetal introspection start [NODE UUID]
$ ironic node-set-maintenance [NODE UUID] false
```

6.1.3. Manually Tagging the Nodes

After registering and inspecting the hardware of each node, we tag them into specific profiles. These profile tags match our nodes to flavors, and in turn the flavors are assigned to a deployment role. For the Basic Deployment scenario, we tag them manually since there are only two nodes. For a larger number of nodes, use the Automated Health Check (AHC) Tools in the Advanced Deployment Scenario. See Section 6.2.3, “Automatically Tagging Nodes with Automated Health Check (AHC) Tools” for more details about the Automated Health Check (AHC) Tools.

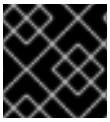
To manually tag a node to a specific profile, add a **`profile`** option to the **`properties/capabilities`** parameter for each node. For example, to tag our two nodes to use a controller profile and a compute profile respectively, use the following commands:

```
$ 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.

6.1.4. Creating Flavors for the Basic Scenario

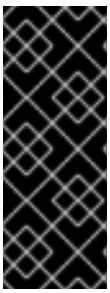
The director also needs a set of hardware profiles, or flavors, for the registered nodes. In this scenario, we'll create a profile each for the Compute and Controller nodes.

```
$ openstack flavor create --id auto --ram 6144 --disk 40 --vcpus 4 control
$ openstack flavor create --id auto --ram 6144 --disk 40 --vcpus 4 compute
```

This creates two flavors for your nodes: **control** and **compute**. We also set the additional properties for each flavor.

```
$ openstack flavor set --property "cpu_arch"="x86_64" --property
"capabilities:boot_option"="local" --property
"capabilities:profile"="compute" compute
$ openstack flavor set --property "cpu_arch"="x86_64" --property
"capabilities:boot_option"="local" --property
"capabilities:profile"="control" control
```

The **capabilities:boot_option** sets the boot mode for the flavor and the **capabilities:profile** defines the profile to use. This links to the same tag on each respective node tagged in Section 6.1.3, “Manually Tagging the Nodes”.



IMPORTANT

Unused roles also require a default flavor named **baremetal**. Create this flavor if it does not exist:

```
$ openstack flavor create --id auto --ram 4096 --disk 40 --
vcpus 1 baremetal
```

6.1.5. 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 Heat template collection.

The Heat template collection contains a set of environment files in **/usr/share/openstack-tripleo-heat-templates/environments/**. These are environment templates to 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, we 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 backend 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

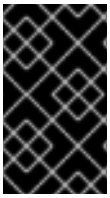
The NFS mount options for the image storage.

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

```
parameters:
  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.

6.1.6. Isolating the External Network

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 scenario uses two separate networks:

- Network 1 - Provisioning network. The Internal API, Storage, Storage Management, and Tenant networks use this network too.
- Network 2 - External network. This network will use a dedicated interface for connecting outside of the Overcloud.

The following sections show how to create Heat templates to isolate the External network from the rest of the services. For more examples of network configuration, see Appendix F, *Network Interface Template Examples*.

6.1.6.1. 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 Chapter 5, *Understanding 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.

For the Basic Overcloud scenario, we use the default single NIC example configuration. Copy the default configuration directory into the **stack** user's home directory as **nic-configs**.

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

This creates a local set of Heat templates that define a single network interface configuration the External network uses. Each template contains the standard **parameters**, **resources**, and **output** sections. For our purposes, we 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 our 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. A bond joins two or more **interfaces** together to help with redundancy and increase bandwidth.

```
            - type: ovs_bond
              name: bond1
```

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

ovs_bridge

Defines a bridge in Open vSwitch. A bridge connects multiple **`interface`**, **`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}
```

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

For the Basic Scenario, modify each interface template to move the External network to **`nic2`**. This ensures we use the second network interface on each node for the External network. For example, for the **`templates/nic-configs/controller.yaml`** template:

```
        network_config:
          - type: ovs_bridge
            name: {get_input: bridge_name}
            use_dhcp: true
            members:
              - type: interface
                name: nic1
                # force the MAC address of the bridge to this interface
                primary: true
              - type: vlan
                vlan_id: {get_param: InternalApiNetworkVlanID}
                addresses:
                  - ip_netmask: {get_param: InternalApiIpSubnet}
              - type: vlan
                vlan_id: {get_param: StorageNetworkVlanID}
                addresses:
                  - ip_netmask: {get_param: StorageIpSubnet}
              - type: vlan
                vlan_id: {get_param: StorageMgmtNetworkVlanID}
                addresses:
                  - ip_netmask: {get_param: StorageMgmtIpSubnet}
```

```
            - type: vlan
              vlan_id: {get_param: TenantNetworkVlanID}
              addresses:
                - ip_netmask: {get_param: TenantIpSubnet}
          - type: interface
            name: nic2
            addresses:
              - ip_netmask: {get_param: ExternalIpSubnet}
            routes:
              - ip_netmask: 0.0.0.0/0
                next_hop: {get_param: ExternalInterfaceDefaultRoute}
```

The above example creates a new interface (**nic2**) and reassigns the External network addresses and routes to the new interface.

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. We define these in an environment file we create specifically for our 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 used interfaces from **ovs_bridge** devices and disable the DHCP and default route settings:

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

6.1.6.2. Creating a Basic Overcloud Network Environment Template

The network environment file describes the Overcloud's network environment and points to the network interface configuration files from the previous section. We define the subnets for our network along with IP address ranges. We customize these values for the local environment.

This scenario uses the following network environment file saved as **/home/stack/templates/network-environment.yaml**:

```
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:
  ExternalNetCidr: 10.1.1.0/24
  ExternalAllocationPools: [{'start': '10.1.1.2', 'end': '10.1.1.50'}]
  ExternalNetworkVlanID: 100
  # 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"]
  # Set to "br-ex" if using floating IPs on native VLAN on bridge br-ex
  NeutronExternalNetworkBridge: "''"
```

The **`resource_registry`** section contains links to the network interface templates for each node role. Note that the **`ExternalAllocationPools`** parameter only defines a small range of IP addresses. This is so we can later define a separate range of floating IP addresses.

The **`parameter_defaults`** section contains 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*.

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 our example, we only have IPs from 10.1.1.10 to 10.1.1.50 assign 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.

This scenario only defines the options for the External network. All other traffic types are automatically assigned to the Provisioning network.

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 used.

6.1.7. Creating the Basic Overcloud

The final stage in creating your OpenStack environment is to run the necessary commands that create it. The default plan installs one Controller node and one Compute node.

NOTE

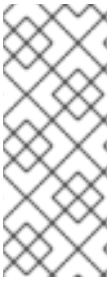
The Red Hat Customer Portal contains a lab to help validate your configuration before creating the Overcloud. This lab is available at https://access.redhat.com/labs/ospec/ and instructions for this lab are available at https://access.redhat.com/labsinfo/ospec.

Run the following command to start the Basic Overcloud creation:

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

This command contains the following additional options:

- **`--templates`** - Creates the Overcloud using the Heat template collection located 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 plan. In this case, it is an environment file that initializes network isolation configuration.
- **`-e /home/stack/templates/network-environment.yaml`** - The **`-e`** option adds an additional environment file to the Overcloud plan. In this case, it is the network environment file we created from Section 6.1.6.2, “Creating a Basic Overcloud Network Environment Template”.
- **`-e /home/stack/templates/storage-environment.yaml`** - The **`-e`** option adds an additional environment file to the Overcloud plan. In this case, it is the storage environment file we created from Section 6.1.5, “Configuring NFS Storage”.
- **`--control-flavor control`** - Use a specific flavor for the Controller nodes.
- **`--compute-flavor compute`** - Use a specific flavor for the Compute nodes.
- **`--ntp-server pool.ntp.org`** - Use an NTP server for time synchronization. This is useful for keeping the Controller node cluster in synchronization.
- **`--neutron-network-type vxlan`** - Use Virtual Extensible LAN (VXLAN) for the Neutron networking in the Overcloud.
- **`--neutron-tunnel-types vxlan`** - Use Virtual Extensible LAN (VXLAN) for Neutron tunneling in the Overcloud.



NOTE

For a full list of options, run:

```
$ openstack help overcloud deploy
```

See also Appendix I, *Deployment Parameters* for parameter examples.

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.

WARNING

Any environment files added to the Overcloud using the **-e** option become part of your Overcloud's stack definition. The director requires these environment files for re-deployment and post-deployment functions in Chapter 7, *Performing Tasks after Overcloud Creation*. Failure to include these files can result in damage to your Overcloud.

If you aim to later modify the Overcloud configuration, modify parameters in the custom environment files and Heat templates, then run the **openstack overcloud deploy** command again. 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.

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.

6.1.8. Accessing the Basic Overcloud

The director generates a file to configure and authenticate interactions with your Overcloud from the Undercloud. The director saves this file, **overcloudrc**, in your **stack** user's home directory. 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
```

6.1.9. Completing the Basic Overcloud

This concludes the creation of the Basic Overcloud. For post-creation functions, see Chapter 7, *Performing Tasks after Overcloud Creation*.

6.2. ADVANCED SCENARIO: CREATING A LARGE OVERCLOUD WITH CEPH STORAGE NODES

This scenario creates a large enterprise-level OpenStack Platform environment with Red Hat Ceph Storage nodes. This scenario consists of nine nodes in the Overcloud:

- Three Controller nodes with high availability
- Three Compute nodes
- Three Red Hat Ceph Storage nodes in a cluster

All machines are bare metal systems using IPMI for power management. This scenario aims to demonstrate the director's ability to create an production-level Red Hat Enterprise Linux OpenStack Platform environment that can scale Compute nodes in the future. This scenario uses the command line tools to demonstrate some of the advanced features of the director, including using the Automated Health Check Tools for role matching, and advanced network isolation.

Workflow

1. Create a node definition template and register blank nodes in the director.
2. Inspect hardware and benchmark all nodes.
3. Use the Automated Health Check (AHC) Tools to define policies that automatically tag nodes into roles.
4. Create flavors and tag them into roles.
5. Use an environment file to configure Ceph Storage.
6. Create Heat templates to isolate all networks.
7. Create the Overcloud environment using the default Heat template collection and the additional network isolation templates.
8. Add fencing information for each Controller node in the high-availability cluster.

Requirements

- The director node created in Chapter 3, *Installing the Undercloud*
- Nine bare metal machines. These machines must comply with the requirements set for the Controller, Compute, and Ceph Storage nodes. For these requirements, see:
  - Section 2.4.2, “Controller Node Requirements”
  - Section 2.4.1, “Compute Node Requirements”
  - Section 2.4.3, “Ceph Storage Node Requirements”

  These nodes do not require an operating system because 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 this example, we use 192.0.2.0/24 as the Provisioning subnet with the following IP address assignments:

  Table 6.3. Provisioning Network IP Assignments

Node Name	IP Address	MAC Address	IPMI IP Address
Director	192.0.2.1	aa:aa:aa:aa:aa:aa	
Controller 1	DHCP defined	b1:b1:b1:b1:b1:b1	192.0.2.205
Controller 2	DHCP defined	b2:b2:b2:b2:b2:b2	192.0.2.206
Controller 3	DHCP defined	b3:b3:b3:b3:b3:b3	192.0.2.207
Compute 1	DHCP defined	c1:c1:c1:c1:c1:c1	192.0.2.208
Compute 2	DHCP defined	c2:c2:c2:c2:c2:c2	192.0.2.209
Compute 3	DHCP defined	c3:c3:c3:c3:c3:c3	192.0.2.210
Ceph 1	DHCP defined	d1:d1:d1:d1:d1:d1	192.0.2.211
Ceph 2	DHCP defined	d2:d2:d2:d2:d2:d2	192.0.2.212
Ceph 3	DHCP defined	d3:d3:d3:d3:d3:d3	192.0.2.213

- Each Overcloud node uses the remaining two network interfaces in a bond to serve networks in tagged VLANs. The following network assignments apply to this bond:

Table 6.4. 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
External / Floating IP	10.1.1.0/24	100

6.2.1. Registering Nodes for the Advanced Overcloud

In this section, we create a node definition template. This file (**`instackenv.json`**) is a JSON format file and contains the hardware and power management details for our nine nodes.

This template uses the following attributes:

mac

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

pm_type

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

pm_user, pm_password

The IPMI username and password.

pm_addr

The IP address of the IPMI device.

cpu

The number of CPUs on the node.

memory

The amount of memory in MB.

disk

The size of the hard disk in GB.

arch

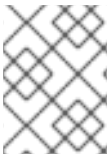
The system architecture.

For example:

```
{
    "nodes":[
        {
            "mac":[
                "b1:b1:b1:b1:b1:b1"
            ],
            "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":[
                "b2:b2:b2:b2:b2:b2"
            ],
            "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"
        },
        {
            "mac":[
                "b3:b3:b3:b3:b3:b3"
            ],
            "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":[
                "c1:c1:c1:c1:c1:c1"
            ],
            "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"
        },
        {
            "mac":[
                "c2:c2:c2:c2:c2:c2"
            ],
            "cpu":"4",
            "memory":"6144",
            "disk":"40",
            "arch":"x86_64",
            "pm_type":"pxe_ipmitool",
            "pm_user":"admin",
            "pm_password":"p@55w0rd!",
            "pm_addr":"192.0.2.209"
        },
        {
            "mac":[
                "c3:c3:c3:c3:c3:c3"
            ],
            "cpu":"4",
            "memory":"6144",
            "disk":"40",
            "arch":"x86_64",
            "pm_type":"pxe_ipmitool",
            "pm_user":"admin",
            "pm_password":"p@55w0rd!",
            "pm_addr":"192.0.2.210"
        },
        {
```

```
            "mac":[
                "d1:d1:d1:d1:d1:d1"
            ],
            "cpu":"4",
            "memory":"6144",
            "disk":"40",
            "arch":"x86_64",
            "pm_type":"pxe_ipmitool",
            "pm_user":"admin",
            "pm_password":"p@55w0rd!",
            "pm_addr":"192.0.2.211"
        },
        {
            "mac":[
                "d2:d2:d2:d2:d2:d2"
            ],
            "cpu":"4",
            "memory":"6144",
            "disk":"40",
            "arch":"x86_64",
            "pm_type":"pxe_ipmitool",
            "pm_user":"admin",
            "pm_password":"p@55w0rd!",
            "pm_addr":"192.0.2.212"
        },
        {
            "mac":[
                "d3:d3:d3:d3:d3:d3"
            ],
            "cpu":"4",
            "memory":"6144",
            "disk":"40",
            "arch":"x86_64",
            "pm_type":"pxe_ipmitool",
            "pm_user":"admin",
            "pm_password":"p@55w0rd!",
            "pm_addr":"192.0.2.213"
        }
    ]
}
```



NOTE

For more supported power management types and their options, see Appendix C, *Power Management Drivers*.

After creating the template, save the file to the **stack** user's home directory as **instackenv.json**, then import it into the director. Use the following command to accomplish this:

```
$ 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 using the following command:

```
$ openstack baremetal list
```

6.2.2. Inspecting the Hardware of Nodes

After registering the nodes, we inspect the hardware attribute of each node. This scenario also benchmarks the node for use with the Automated Health Check (AHC) Tools, which we use to automatically tag nodes into deployment profiles. These profile tags match our nodes to flavors, and in turn the flavors are assigned to a deployment role.

IMPORTANT

The benchmarking feature requires the **discovery_runbench** option set to true when initially configuring the director (see Section 3.6, “Configuring the Director”).

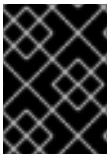
If you need to enable benchmarking after installing the director, edit the **/httpboot/discoverd.ipxe** and set the **RUNBENCH** kernel parameter to **1**.

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-discoverd -u openstack-ironic-
discoverd-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 maintenance mode, perform the introspection, then revert the node out of maintenance mode:

```
$ ironic node-set-maintenance [NODE UUID] true
$ openstack baremetal introspection start [NODE UUID]
$ ironic node-set-maintenance [NODE UUID] false
```

6.2.3. Automatically Tagging Nodes with Automated Health Check (AHC) Tools

Once the discovery process completes its benchmark tests, you can generate a set of reports to identify and isolate underperforming or unstable nodes from use in the Overcloud. This section examines how to generate these reports and create policies to automatically tag nodes into certain roles.

Install the following Automated Health Check (AHC) tools using the following command:

-

```
$ sudo yum install -y ahc-tools
```

The package contains two tools:

- **ahc-report**, which provides reports from the benchmark tests.
- **ahc-match**, which tags nodes into specific roles based on policies.

IMPORTANT

These tools require credentials for Ironic and Swift set in the **/etc/ahc-tools/ahc-tools.conf** file. These are the same credentials in **/etc/ironic-discoverd/discoverd.conf**. Use the following commands to copy and tailor the configuration file for **/etc/ahc-tools/ahc-tools.conf**:

```
$ sudo -i
# mkdir /etc/ahc-tools
# sed 's/\[discoverd/\[ironic/' /etc/ironic-
discoverd/discoverd.conf > /etc/ahc-tools/ahc-tools.conf
# chmod 0600 /etc/ahc-tools/ahc-tools.conf
# exit
```

6.2.3.1. ahc-report

The **ahc-report** script produces various reports about your nodes. To view a full report, use the **--full** option.

```
$ sudo ahc-report --full
```

The **ahc-report** command can also focus on specific parts of a report. For example, use the **--categories** to categorize nodes based on their hardware (processors, network interfaces, firmware, memory, and various hardware controllers). This also groups these nodes together with similar hardware profiles. For example, the **Processors** section for our two example nodes might list the following:

```
######################
##### Processors #####
2 identical systems :
[u'7F8831F1-0D81-464E-A767-7577DF49AAA5', u'7884BC95-6EF8-4447-BDE5-
D19561718B29']
[(u'cpu', u'logical', u'number', u'4'),
 (u'cpu', u'physical', u'number', u'4'),
 (u'cpu',
  u'physical_0',
  u'flags',
  u'fpu fpu_exception wp de pse tsc msr pae mce cx8 apic sep mtrr pge mca
cmov pse36 clflush mmx fxsr sse sse2 syscall nx x86-64 rep_good nopl pni
cx16 hypervisor lahf_lm'),
 (u'cpu', u'physical_0', u'frequency', u'2000000000'),
 (u'cpu', u'physical_0', u'physid', u'0'),
 (u'cpu', u'physical_0', u'product', u'Intel(R) Xeon(TM) CPU       E3-
1271v3 @ 3.6GHz'),
 (u'cpu', u'physical_0', u'vendor', u'GenuineIntel'),
 (u'cpu',
```

```
  u'physical_1',
  u'flags',
  u'fpu fpu_exception wp de pse tsc msr pae mce cx8 apic sep mtrr pge mca
cmov pse36 clflush mmx fxsr sse sse2 syscall nx x86-64 rep_good nopl pni
cx16 hypervisor lahf_lm'),
 (u'cpu', u'physical_0', u'frequency', u'2000000000'),
 (u'cpu', u'physical_0', u'physid', u'0'),
 (u'cpu', u'physical_0', u'product', u'Intel(R) Xeon(TM) CPU       E3-
1271v3 @ 3.6GHz'),
 (u'cpu', u'physical_0', u'vendor', u'GenuineIntel')
 ...
 ]
```

The **ahc-report** tool also identifies the outliers in your node collection. Use the **--outliers** switch to enable this:

```
$ sudo ahc-report --outliers

Group 0 : Checking logical disks perf
standalone_randread_4k_KBps   : INFO    : sda   : Group performance :
min=45296.00, mean=53604.67, max=67923.00, stddev=12453.21
standalone_randread_4k_KBps   : ERROR   : sda   : Group's variance is too
important :   23.23% of 53604.67 whereas limit is set to 15.00%
standalone_randread_4k_KBps   : ERROR   : sda   : Group performance :
UNSTABLE
standalone_read_1M_IOps       : INFO    : sda   : Group performance : min=
1199.00, mean= 1259.00, max= 1357.00, stddev=   85.58
standalone_read_1M_IOps       : INFO    : sda   : Group performance =
1259.00   : CONSISTENT
standalone_randread_4k_IOps   : INFO    : sda   : Group performance :
min=11320.00, mean=13397.33, max=16977.00, stddev= 3113.39
standalone_randread_4k_IOps   : ERROR   : sda   : Group's variance is too
important :   23.24% of 13397.33 whereas limit is set to 15.00%
standalone_randread_4k_IOps   : ERROR   : sda   : Group performance :
UNSTABLE
standalone_read_1M_KBps       : INFO    : sda   : Group performance :
min=1231155.00, mean=1292799.67, max=1393152.00, stddev=87661.11
standalone_read_1M_KBps       : INFO    : sda   : Group performance =
1292799.67   : CONSISTENT

...
```

In the example above, **ahc-report** marked the **standalone_randread_4k_KBps** and **standalone_randread_4k_IOps** disk metrics as unstable due to the standard deviation of all nodes being higher than the allowable threshold. In our example, this could happen if our two nodes have a significant difference in disk transfer rates.

It is useful to determine outliers in your node collection because you can assign high performance nodes for more suitable tasks. For example, nodes with better disk transfer rates make better storage nodes, while nodes with better memory performance might make better Compute nodes. Once you have identified hardware performance of each node, create a set of policies and use the **ahc-match** command to assign nodes to specific roles.

6.2.3.2. ahc-match

The **ahc-match** command applies a set of policies to your Overcloud plan to help assign nodes to certain roles. Prior to using this command, create a set of policies to match suitable nodes to roles.

The **ahc-tools** package installs a set of policy files under **/etc/ahc-tools/edeploy**. This includes:

- **state** - The state file, which outlines the number of nodes for each role.
- **compute.specs**, **control.specs** - Policy files for matching Compute and Controller nodes.
- **compute.cmdb.sample**, **control.cmdb.sample** - Sample Configuration Management Database (CMDB) files, which contain key/value settings for RAID and BIOS ready-state configuration (Dell DRAC only).

State File
The **state** file indicates the number of nodes for each role. The default configuration file shows:

```
[('control', '1'), ('compute', '*')]
```

This means the **ahc-match** assigns one control node and any number of compute nodes. For this scenario, edit this file:

```
[('control', '3'), ('ceph-storage', '3'), ('compute', '*')]
```

This matches three Controller nodes, three Red Hat Ceph Storage nodes, and an unlimited number of Compute nodes.

Policy Files
The **compute.specs** and **control.specs** files list the assignment rules for each respective role. The file contents is a tuple format, such as:

```
[
 ('cpu', 'logical', 'number', 'ge(2)'),
 ('disk', '$disk', 'size', 'gt(4)'),
 ('network', '$eth', 'ipv4', 'network(192.0.2.0/24)'),
 ('memory', 'total', 'size', 'ge(4294967296)'),
]
```

This provides a way to define assignments rules based on hardware parameters. For a full reference of all parameters available, see Appendix D, *Automated Health Check (AHC) Tools Parameters*.

The policy files also use a set of helper functions to match value ranges. These functions are

- **network()** - The network interface is in the specified network.
- **gt()**, **ge()** - Greater than (or equal).
- **lt()**, **le()** - Lower than (or equal).
- **in()** - The item to match shall be in a specified set.
- **regexp()** - Match a regular expression.
- **or()**, **and()**, **not()** - Boolean functions. **or()** and **and()** take two parameters and **not()** one parameter.

For example, this scenario uses the **standalone_randread_4k_KBps** and **standalone_randread_4k_IOps** values from Section 6.2.3.1, “ahc-report” to limit the Controller role to node with disk access rates higher than the average rate. The rules for each would be:

```
[
 ('disk', '$disk', 'standalone_randread_4k_KBps', 'gt(53604)'),
 ('disk', '$disk', 'standalone_randread_4k_IOps', 'gt(13397)')
]
```

You can also create additional policy profiles for other roles. For example, create a **ceph-storage.spec** for a profile specifically for Red Hat Ceph Storage. Ensure these new filenames (without extension) are included in the **state** file.

Ready-State Files (Dell DRAC only)
The ready-state configuration prepares bare metal resources for deployment. This includes BIOS and RAID configuration for predefined profiles.

To define a BIOS setting, define a JSON tuple that define each setting and target value for the **bios_settings** key. For example:

```
[
  {
    'bios_settings': {'ProcVirtualization': 'Enabled', 'ProcCores': 4}
  }
]
```

You configure the RAID configuration in two ways:

- **List the IDs of the physical disks** - Provide a list of physical disk IDs using the following attributes: **controller**, **size_gb**, **raid_level** and the list of **physical_disks**. **controller** should be the FQDD of the RAID controller that the DRAC assigns. Similarly, the list of **physical_disks** should be the FQDDs of physical disks the DRAC card assigns.

  ```
  [
    {
      'logical_disks': [
        {'controller': 'RAID.Integrated.1-1',
          'size_gb': 100,
          'physical_disks': [
            'Disk.Bay.0:Enclosure.Internal.0-1:RAID.Integrated.1-1',
            'Disk.Bay.1:Enclosure.Internal.0-1:RAID.Integrated.1-1',
            'Disk.Bay.2:Enclosure.Internal.0-1:RAID.Integrated.1-1'],
          'raid_level': '5'},
      ]
    }
  ]
  ```

- **Let Ironic assign physical disks to the RAID volume** - The following attributes are required: **controller**, **size_gb**, **raid_level** and the **number_of_physical_disks**. **controller** should be the FQDD of the RAID controller the DRAC card assigns.

  ```
  [
    {
      'logical_disks': [
  ```

```
      {'controller': 'RAID.Integrated.1-1',
        'size_gb': 50,
        'raid_level': '1',
        'number_of_physical_disks': 2},
    ]
  }
]
```

Running the Matching Tool
After defining your rules, run the **`ahc-match`** tool to assign your nodes.

```
$ sudo ahc-match
```

This matches all nodes to the roles defined in **`/etc/ahc-tools/edeploy/state`**. When a node matches a role, **`ahc-match`** adds the role to the node in Ironic as a capability.

```
$ ironic node-show b73fb5fa-1a2c-49c6-b38e-8de41e3c0532 | grep properties
-A2
| properties    | {u'memory_mb': u'6144', u'cpu_arch': u'x86_64',
u'local_gb': u'40',      |
|               | u'cpus': u'4', u'capabilities':
u'profile:control,boot_option:local'}    |
| instance_uuid | None
|
```

The director uses this **`profile`** tag from each node to match to roles and flavors with the same tag.

If you also configured RAID and BIOS ready-state settings, run the following command to configure these on each node:

```
$ instack-ironic-deployment --configure-nodes
```

6.2.4. Creating Hardware Profiles

The director also needs a set of hardware profiles, or flavors, for the registered nodes. In this scenario, we'll create a profile each for the Compute, Controller, and Ceph Storage nodes:

```
$ openstack flavor create --id auto --ram 6144 --disk 40 --vcpus 4 control
$ openstack flavor create --id auto --ram 6144 --disk 40 --vcpus 4 compute
$ openstack flavor create --id auto --ram 6144 --disk 40 --vcpus 4 ceph-
storage
```

IMPORTANT

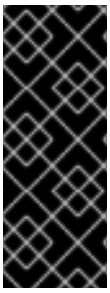
The values for the three flavors in this scenario are for example purposes only. Use the specifications for your hardware identified with the AHC Tools.

This creates three flavors for your nodes. We also set the additional properties for each flavor.

```
$ openstack flavor set --property "cpu_arch"="x86_64" --property
"capabilities:boot_option"="local" --property
"capabilities:profile"="compute" compute
```

```
$ openstack flavor set --property "cpu_arch"="x86_64" --property
"capabilities:boot_option"="local" --property
"capabilities:profile"="control" control
$ openstack flavor set --property "cpu_arch"="x86_64" --property
"capabilities:boot_option"="local" --property
"capabilities:profile"="ceph-storage" ceph-storage
```

The **capabilities:boot_option** sets the boot mode for the flavor and the **capabilities:profile** defines the profile to use.



IMPORTANT

Unused roles also require a default flavor named **baremetal**. Create this flavor if it does not exist:

```
$ openstack flavor create --id auto --ram 4096 --disk 40 --
vcpus 1 baremetal
```

6.2.5. Configuring Ceph Storage

This section describes installing and configuring Red Hat Ceph Storage using Director for use with OpenStack. The installation and configuration process is based on a combination of Heat Templates and Puppet configuration.

The Overcloud image already contains the Ceph Storage software and the necessary Puppet modules to automatically configure both the Ceph OSD nodes and the Ceph Monitor on Controller clusters. The Overcloud's Heat template collection also contains the necessary configuration to enable your Ceph Storage configuration.

The Ceph Storage cluster might require some minor configuration, specifically the disk layout on the Ceph Storage nodes. To pass this information, copy the **storage-environment.yaml** environment file to your **stack** user's **templates** directory.

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

Modify the following options in the copy of **storage-environment.yaml**:

CinderEnableIscsiBackend

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

CinderEnableRbdBackend

Enables the Ceph Storage backend. Set to **true**.

CinderEnableNfsBackend

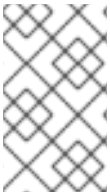
Enables the NFS backend. Set to **false**.

NovaEnableRbdBackend

Enables Ceph Storage for Nova ephemeral storage. Set to **true**.

GlanceBackend

Define the backend to use for Glance. Set to **rbd** to use Ceph Storage for images.



NOTE

The **storage-environment.yaml** also contains some options to configure Ceph Storage directly through Heat. However, these options are not necessary in this scenario since the director creates these nodes and automatically defines the configuration values.

Add an additional section to this environment file that contains the following:

```
parameter_defaults:
  ExtraConfig:
    ceph::profile::params::osds:
```

This adds extra Hiera data to the Overcloud, which is used in the Puppet configuration. For more information, see Section 10.4, “Customizing Puppet Configuration Data”.

Use **ceph::profile::params::osds** parameter to map the relevant journal partitions and disks. For example, a Ceph node with four disks might have the following assignments:

- **/dev/sda** - The root disk containing the Overcloud image
- **/dev/sdb** - The disk containing the journal partitions. This is usually a solid state disk (SSD) to aid with system performance.
- **/dev/sdc** and **/dev/sdd** - The OSD disks

For this example, the mapping might contain the following:

```
    ceph::profile::params::osds:
      '/dev/sdc':
        journal: '/dev/sdb'
      '/dev/sdd':
        journal: '/dev/sdb'
```

If you do not want a separate disk for journals, use co-located journals on the OSD disks. Pass a blank value to the **journal** parameters:

```
    ceph::profile::params::osds:
        '/dev/sdb': {}
        '/dev/sdc': {}
        '/dev/sdd': {}
```

The **storage-environment.yaml** file's options should look similar to the following:

```
parameters:
  CinderEnableIscsiBackend: false
  CinderEnableRbdBackend: true
  CinderEnableNfsBackend: false
  NovaEnableRbdBackend: true

parameter_defaults:
  ExtraConfig:
```

```
ceph::profile::params::osds:
  '/dev/sdc':
    journal: '/dev/sdb'
  '/dev/sdd':
    journal: '/dev/sdb'
```

After completing these modifications, save the **storage-environment.yaml** so that when we deploy the Overcloud, the Ceph Storage nodes will use our disk mapping and custom settings. We include this file in our deployment to initiate our storage requirements.

IMPORTANT

The Ceph Storage OSDs should be unpartitioned disks with 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
```

6.2.6. Isolating all Networks into VLANs

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 scenario 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 - External and Floating IP (mapped after Overcloud creation)

The following sections show how to create Heat templates to isolate all network types. For more examples of network configuration, see Appendix F, *Network Interface Template Examples*.

6.2.6.1. 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 Chapter 5, *Understanding 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.

For the Advanced Overcloud scenario, we 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 our purposes, we 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 our 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. A bond joins two or more **interfaces** together to help with redundancy and increase bandwidth.

```
            - type: ovs_bond
              name: bond1
              members:
              - type: interface
```

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

ovs_bridge

Defines a bridge in Open vSwitch. A bridge connects multiple **`interface`**, **`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_bridge

Defines a Linux bridge. Similar to an Open vSwitch bridge, it connects multiple **`interface`**, **`bond`** and **`vlan`** objects together.

```
          - type: linux_bridge
            name: bridge1
            members:
            - type: interface
              name: nic1
              primary: true
          - type: vlan
            device: bridge1
            vlan_id: {get_param: ExternalNetworkVlanID}
            addresses:
              - ip_netmask: {get_param: ExternalIpSubnet}
```

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

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

```
    network_config:
      - type: interface
        name: nic1
        use_dhcp: false
        addresses:
```

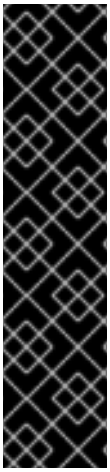
```
                - ip_netmask:
                    list_join:
                      - '/'
                      - - {get_param: ControlPlaneIp}
                        - {get_param: ControlPlaneSubnetCidr}
              routes:
                - ip_netmask: 169.254.169.254/32
                  next_hop: {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:
                    - ip_netmask: 0.0.0.0/0
                      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}
```

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.

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. We define these in an environment file we create specifically for our 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
```

6.2.6.2. Creating an Advanced Overcloud 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. We define the subnets and VLANs for our network along with IP address ranges. We customize these values for the local environment.

This scenario uses the following network environment file saved as **`/home/stack/templates/network-environment.yaml`**:

```
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
  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'}]
```

```
  # 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
  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"
```

The **resource_registry** section contains links to the network interface templates for each node role.

The **parameter_defaults** section contains 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*.

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 our 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 our example, we only have IPs from 10.1.1.10 to 10.1.1.50 assign 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, *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 used.

6.2.6.3. 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, we can reassign the Storage Management network services to the Storage Network by modifying the highlighted sections:

```
...

parameter_defaults:

  ServiceNetMap:
    NeutronTenantNetwork: tenant
    CeilometerApiNetwork: internal_api
    MongoDbNetwork: internal_api
    CinderApiNetwork: internal_api
    CinderIscsiNetwork: storage
    GlanceApiNetwork: storage
    GlanceRegistryNetwork: internal_api
    KeystoneAdminApiNetwork: internal_api
    KeystonePublicApiNetwork: internal_api
    NeutronApiNetwork: internal_api
    HeatApiNetwork: internal_api
    NovaApiNetwork: internal_api
    NovaMetadataNetwork: internal_api
    NovaVncProxyNetwork: internal_api
    SwiftMgmtNetwork: storage_mgmt
    SwiftProxyNetwork: storage
    HorizonNetwork: internal_api
    MemcachedNetwork: internal_api
    RabbitMqNetwork: internal_api
    RedisNetwork: internal_api
    MysqlNetwork: internal_api
    CephClusterNetwork: storage_mgmt
    CephPublicNetwork: storage
    # Define which network will be used for hostname resolution
    ControllerHostnameResolveNetwork: internal_api
    ComputeHostnameResolveNetwork: internal_api
    BlockStorageHostnameResolveNetwork: internal_api
    ObjectStorageHostnameResolveNetwork: internal_api
    CephStorageHostnameResolveNetwork: storage
```

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.

6.2.7. Enabling SSL/TLS on the Overcloud

The Overcloud uses unencrypted endpoints for its services by default. This means the Overcloud configuration requires an additional environment file to enable SSL/TLS for its endpoints.

This process requires network isolation to define the endpoints for the Public API. See Section 6.2.6, “Isolating all Networks into VLANs” for instruction on network isolation.

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

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:

parameter_defaults:

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 B, *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.

resource_registry:

OS::TripleO::NodeTLSData:

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

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

Injecting a Root Certificate

If using a self-signed certificate or the certificate signer is not in the default trust store on the Overcloud image, inject the certificate 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/.
```

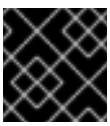
Edit this file and make the following changes for these parameters:

parameter_defaults:

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.

resource_registry:

OS::TripleO::NodeTLSCAData:

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

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

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:

parameter_defaults:

CloudName:

The DNS hostname for the Overcloud endpoints.

DnsServers:

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

The following is an example of the contents for this file:

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

Adding Environment Files During Overcloud Creation

The deployment command (**openstack overcloud deploy**) in Section 6.2.9, "Creating the Advanced 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
```

6.2.8. 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.

Method 1 - Command Line

The deployment command (**openstack overcloud deploy**) uses a set of options to define your

registration details. The table in Appendix I, *Deployment Parameters* contains these options and their descriptions. Include these options when running the deployment command in Section 6.2.9, "Creating the Advanced 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 [...]
```

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 either **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 6 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.

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

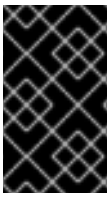
The machine name. Leave this as blank to use the hostname of the node.

rhel_reg_force

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

The deployment command (**`openstack overcloud deploy`**) in Section 6.2.9, “Creating the Advanced 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 10.2, “Customizing Overcloud Pre-Configuration” for more information.

6.2.9. Creating the Advanced Overcloud

The final stage in creating your OpenStack environment is to run the necessary commands to create it. The command options we use install three Controller nodes, three Compute nodes, and three Ceph Storage nodes.

NOTE

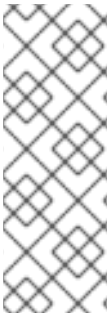
The Red Hat Customer Portal contains a lab to help validate your configuration before creating the Overcloud. This lab is available at https://access.redhat.com/labs/ospec/ and instructions for this lab are available at https://access.redhat.com/labsinfo/ospec.

Run the following command to start the Advanced Overcloud creation:

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

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 6.2.6.2, “Creating an Advanced Overcloud 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.
- **`--neutron-network-type vxlan`** - Use Virtual Extensible LAN (VXLAN) for the Neutron networking in the Overcloud.
- **`--neutron-tunnel-types vxlan`** - Use Virtual Extensible LAN (VXLAN) for Neutron tunneling in the Overcloud.



NOTE

For a full list of options, run:

```
$ openstack help overcloud deploy
```

See also Appendix I, *Deployment Parameters* for parameter examples and Section 6.2.8, "Registering the Overcloud" for registration details.

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.

WARNING

Any environment files added to the Overcloud using the **-e** option become part of your Overcloud's stack definition. The director requires these environment files for re-deployment and post-deployment functions in Chapter 7, *Performing Tasks after Overcloud Creation*. Failure to include these files can result in damage to your Overcloud.

If you aim to later modify the Overcloud configuration, modify parameters in the custom environment files and Heat templates, then run the **openstack overcloud deploy** command again. 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.

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 \
  --neutron-network-type vxlan \
  --neutron-tunnel-types vxlan \
  --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.

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.

6.2.10. Accessing the Advanced 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
```

6.2.11. 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 a tool called 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 various power management devices the director supports. This includes:

Table 6.5. Fence Agents

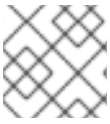
Device	Type
fence_ipmilan	The Intelligent Platform Management Interface (IPMI)
fence_idrac, **fence_drac5**	Dell Remote Access Controller (DRAC)
fence_ilo	Integrated Lights-Out (iLO)
fence_ucs	Cisco UCS - For more information, see Configuring Cisco Unified Computing System (UCS) Fencing on an OpenStack 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 in 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 1:

```
$ sudo pcs stonith create my-ipmilan-for-controller01 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-controller01 avoids
overcloud-controller-0
```

For Controller node 2:

```
$ sudo pcs stonith create my-ipmilan-for-controller02 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-controller02 avoids
overcloud-controller-1
```

For Controller node 3:

```
$ sudo pcs stonith create my-ipmilan-for-controller03 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-controller03 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
```

6.2.12. Completing the Advanced Overcloud

This concludes the creation of the Advanced Overcloud. For post-creation functions, see Chapter 7, *Performing Tasks after Overcloud Creation*.

CHAPTER 7. PERFORMING TASKS AFTER OVERCLOUD CREATION

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

7.1. CREATING THE OVERCLOUD TENANT NETWORK

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

```
$ source ~/overcloudrc
$ 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 |
+----------------------+-------------+------------------------------
---------------------+
```

7.2. CREATING THE OVERCLOUD EXTERNAL NETWORK

In the Basic and Advanced Overcloud scenarios, we configured the node interfaces to use the External network. However, we still need to create this network on the Overcloud so that we 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 nova --router:external --provider:network_type flat -
-provider:physical_network datacentre
$ neutron subnet-create --name nova --enable_dhcp=False --allocation-
pool=start=10.1.1.51,end=10.1.1.250 --gateway=10.1.1.1 nova 10.1.1.0/24
```

In this example, we create a network with the name **nova**. The Overcloud requires this specific name for the default floating IP pool. This is also important for the validation tests in Section 7.5, “Validating the Overcloud”.

This command also maps the network to the **datacenter** physical network. As a default, **datacenter** 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 nova --router:external --provider:network_type vlan --provider:physical_network datacentre --provider:segmentation_id 104
$ neutron subnet-create --name nova --enable_dhcp=False --allocation-pool=start=10.1.1.51,end=10.1.1.250 --gateway=10.1.1.1 nova 10.1.1.0/24
```

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

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

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

7.3. CREATING ADDITIONAL FLOATING IP NETWORKS

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

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

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

Create the Floating IP network after creating the Overcloud using the following commands:

```
$ 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
```

7.4. CREATING THE OVERCLOUD PROVIDER NETWORK

A provider network is a network attached physically to a datacenter network existing outside of the deployed Overcloud. This can be an existing infrastructure network or a network that provides external access directly to VMs 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 tp provide DHCP services to the tenant VMs:

```
$ 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
```

7.5. 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
```

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

```
$ mkdir ~/tempest
$ cd ~/tempest
$ /usr/share/openstack-tempest-kilo/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 7.2, “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 output displays each test and the 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_specify_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 and 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
```

7.6. MODIFYING THE OVERCLOUD ENVIRONMENT

Sometimes you might aim to modify the Overcloud to add additional features or change the way it operates. To modify the Overcloud, make modifications to your custom environment files and Heat templates, then rerun the **openstack overcloud deploy** command from your initial Overcloud creation. For example, if you created an Overcloud using Section 6.2.9, “Creating the Advanced 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 --neutron-network-type vxlan --neutron-tunnel-types vxlan
```

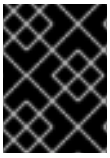
The director checks the **overcloud** stack in Heat and updates each item in the stack with the environtment files and Heat templates. It does not recreate the Overcloud, but rather changes the existing Overcloud.

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

```
$ openstack overcloud deploy --templates -e /usr/share/openstack-tripleo-
```

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

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



IMPORTANT

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

7.7. IMPORTING VIRTUAL MACHINES INTO THE OVERCLOUD

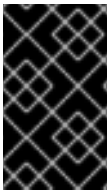
Use the following procedure if you have an existing OpenStack environment and aim to migrate its virtual machines to your Red Hat OpenStack Platform environment.

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.

7.8. 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 7 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-0.8.6-135.el7ost** package and later versions.

Update your environment to use the Heat templates from the **openstack-tripleo-heat-templates-0.8.6-135.el7ost** package or later versions.

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

Procedure 7.1. Migrating Virtual Machines from the Compute Node

1. From the director, source the **overcloudrc** and obtain a list of the current Nova services:

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

2. Disable the **nova-compute** service on the node to migrate.

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

   This prevents new VMs from being scheduled on it.

3. Begin the process of migrating VMs off the node:

```
$ nova host-servers-migrate [hostname]
```

4. The current status of the migration process can be retrieved with the command:

```
$ nova migration-list
```

5. When migration of each VM 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]
```

This migrates all VMs 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
```

7.9. 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. The allow removal of the Overcloud, revert the policy to the original value.

7.10. REMOVING THE OVERCLOUD

The whole Overcloud can be removed when desired.

Procedure 7.2. Removing the Overcloud

1. Delete any existing Overcloud:

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

2. 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 8. 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.

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

Table 8.1. 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.
Cinder Storage Nodes	N	N	
Swift Storage Nodes	N	N	

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.

8.1. ADDING COMPUTE OR CEPH STORAGE 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 6.2.1, “Registering Nodes for the Advanced 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-list
$ ironic node-set-maintenance [NODE UUID] true
$ openstack baremetal introspection start [NODE UUID]
$ ironic node-set-maintenance [NODE UUID] false
```

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'
```

Alternatively, you can automatically tag new nodes into desired roles using the Automated Health Check (AHC) Tools. See Section 6.2.3, “Automatically Tagging Nodes with Automated Health Check (AHC) Tools” for more information.

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 |
+--------------------------------------+------------------------+
```

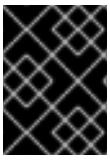
Set these UUIDs for the new node's **deploy_kernel** and **deploy_ramdisk** settings:

```
$ 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.

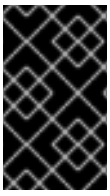


IMPORTANT

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

8.2. 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 7.8, “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 service-list
$ neutron service-delete [openvswitch-service-id]
$ source ~/stack/stackrc
```

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

8.3. 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:

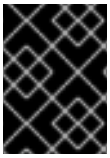
1. Migrate workload off the existing Compute node and shutdown the node. See Section 7.8, “Migrating VMs from an Overcloud Compute Node” for this process.
2. Remove the Compute node from the Overcloud. See Section 8.2, “Removing Compute Nodes” for this process.
3. Scale out the Overcloud with a new Compute node. See Chapter 8, *Scaling the Overcloud* for this process.

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

8.4. 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.

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

8.4.2. 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:

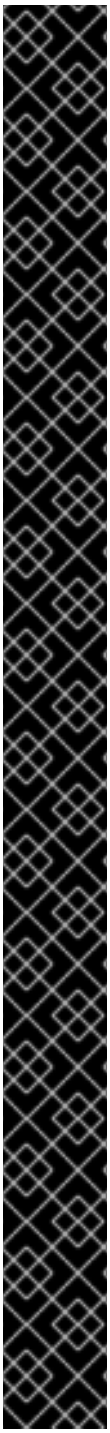
```
[stack@director ~]$ ironic node-set-maintenance da3a8d19-8a59-4e9d-923a-
6a336fe10284 true
```

Tag the new node as 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 Chapter 10, *Creating Custom Configuration* and run the following command on the **`overcloud-without-mergepy.yaml`** file:

```
$ sudo sed -i "s/resource\.0/resource.1/g" ~/templates/my-
overcloud/overcloud-without-mergepy.yaml
```

This changes the node index for the following resources:

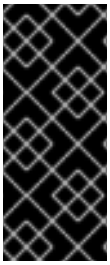
```
ControllerBootstrapNodeConfig:
  type: OS::TripleO::BootstrapNode::SoftwareConfig
    properties:
      bootstrap_nodeid: {get_attr: [Controller,
resource.0.hostname]}
      bootstrap_nodeid_ip: {get_attr: [Controller,
resource.0.ip_address]}
```

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]}
```

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]
```



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 **`-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
```

8.4.3. 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@192.168.201.47 "sudo pcs cluster
localnode remove overcloud-controller-1"
[stack@director] ssh heat-admin@192.168.201.47 "sudo pcs cluster
reload corosync"
[stack@director] ssh heat-admin@192.168.201.46 "sudo pcs cluster
localnode remove overcloud-controller-1"
[stack@director] ssh heat-admin@192.168.201.46 "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@192.168.201.47
[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**:

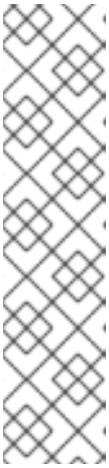
```
tripleo:PRIMARY> rs.remove('192.168.0.45:27017')
```

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. Add the new node to the cluster:

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

9. 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.

10. 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.

11. Start the new Controller node:

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

12. Enable the keystone service on the new node. Copy the **/etc/keystone** directory from a remaining node to the director host:

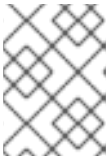
```
[heat-admin@overcloud-controller-0 ~]$ sudo -i
[root@overcloud-controller-0 ~]$ scp -r /etc/keystone
stack@192.168.0.1:~/.
```

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 address. 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"
```

13. 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
```

14. 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 overcloud-controller-3
```

15. Wait until the Keystone service starts on all nodes.

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

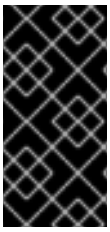
16. Switch the cluster back into maintenance mode:

```
[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.

8.4.4. 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.

Enable fencing if you disabled it during the node replacement. For example, if 192.168.0.47 is the IP address of a running Controller node, use the following command to enable fencing:

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

Exit to the director

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

8.4.5. Finalizing Overcloud Network Agents

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

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

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

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

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

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

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

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

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

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

8.4.6. 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@192.168.201.47
[heat-admin@overcloud-controller-0 ~]$ pcs resource restart openstack-
nova-consoleauth
```

8.4.7. Conclusion

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

8.5. REPLACING CEPH STORAGE NODES

A situation might occur when a Ceph Storage node fails. In this situation, you must ensure to disable and rebalance the faulty node before removing it from the Overcloud to ensure no data loss. This procedure explains the process for replacing a Ceph Storage node.

NOTE

This procedure uses steps from the *Red Hat Ceph Storage Administration Guide* to manually remove Ceph Storage nodes. For more in-depth information about manual removal of Ceph Storage nodes, see Chapter 15. Removing OSDs (Manual) from the *Red Hat Ceph Storage Administration Guide*.

1. Log into either a Controller node or a Ceph Storage node as the **heat-admin** user. The director's **stack** user has an SSH key to access the **heat-admin** user.

2. List the OSD tree and find the OSDs for your node. For example, your node to remove might contain the following OSDs:

```
-2 0.09998     host overcloud-cephstorage-0
0 0.04999          osd.0                        up  1.00000
1.00000
1 0.04999          osd.1                        up  1.00000
1.00000
```

3. Disable the OSDs on the Ceph Storage node. In this case, the OSD IDs are 0 and 1.

```
[heat-admin@overcloud-controller-0 ~]$ sudo ceph osd out 0
[heat-admin@overcloud-controller-0 ~]$ sudo ceph osd out 1
```

The Ceph Storage cluster begins rebalancing. Wait for this process to complete. You can follow the status using the following command:

```
[heat-admin@overcloud-controller-0 ~]$ sudo ceph -w
```

4. Once the Ceph cluster completes rebalancing, log into the faulty Ceph Storage node as the **heat-admin** user and stop the node.

```
[heat-admin@overcloud-cephstorage-0 ~]$ sudo /etc/init.d/ceph stop
osd.0
[heat-admin@overcloud-cephstorage-0 ~]$ sudo /etc/init.d/ceph stop
osd.1
```

5. Remove the Ceph Storage node from the CRUSH map so that it no longer receives data.

```
[heat-admin@overcloud-cephstorage-0 ~]$ sudo ceph osd crush remove
osd.0
[heat-admin@overcloud-cephstorage-0 ~]$ sudo ceph osd crush remove
osd.1
```

6. Remove the OSD authentication key.

```
[heat-admin@overcloud-cephstorage-0 ~]$ sudo ceph auth del osd.0
[heat-admin@overcloud-cephstorage-0 ~]$ sudo ceph auth del osd.1
```

7. Remove the OSD from the cluster.

```
[heat-admin@overcloud-cephstorage-0 ~]$ sudo ceph osd rm 0
[heat-admin@overcloud-cephstorage-0 ~]$ sudo ceph osd rm 1
```

8. Leave the node and return to the director host as the **stack** user.

```
[heat-admin@overcloud-cephstorage-0 ~]$ exit
[stack@director ~]$
```

9. Disable the Ceph Storage node so the director does not reprovision it.

```
[stack@director ~]$ ironic node-list
[stack@director ~]$ ironic node-set-maintenance [UUID] true
```

10. Removing a Ceph Storage node requires an update to the **overcloud** stack in the director using the local template files. First identify the UUID of the Overcloud stack:

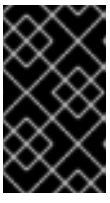
```
$ heat stack-list
```

Identify the UUIDs of the Ceph Storage node 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 again here using the **-e** or **--environment-file** option to avoid making undesired changes to the Overcloud.

Wait until the stack completes its update. Monitor the stack update using the **heat stack-list --show-nested**.

11. Follow the procedure in Section 8.1, “Adding Compute or Ceph Storage Nodes” to add new nodes to the director's node pool and deploy them as Ceph Storage nodes. Use the **--ceph-storage-scale** to define the total number of Ceph Storage nodes in the Overcloud. For example, if you removed a faulty node from a three node cluster and you want to replace it, use **--ceph-storage-scale 3** to return the number of Ceph Storage nodes to its original value:

```
$ openstack overcloud deploy --templates --ceph-storage-scale 3 -e
[ENVIRONMENT_FILES]
```

IMPORTANT

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

The director provisions the new node and updates the entire stack with the new node's details

12. Log into a Controller node as the **heat-admin** user and check the status of the Ceph Storage node. For example:

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

Confirm that the value in the **osdmap** section matches the number of desired nodes in your cluster.

The failed Ceph Storage node has now been replaced with a new node.

CHAPTER 9. 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

9.1. 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
$ nova list
$ ironic node-list
$ heat stack-list
```

9.2. 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
```

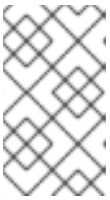
The remaining Controller Nodes in the cluster retain the high availability services during the reboot.

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. Log out of the node, select the next Controller Node to reboot, and repeat this procedure until you have rebooted all Controller Nodes.

9.3. 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
```

9.4. 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

From the undercloud, list all Compute nodes and their UUIDs:

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

2. List all instances on the Compute node:

```
$ nova 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 7.8, “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
$ nova service-enable [hostname] nova-compute
```

4. Select the next node to reboot.

9.5. 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 10. CREATING CUSTOM CONFIGURATION

In some cases, you might want to provide configuration for additional applications that integrate with your Red Hat Enterprise Linux OpenStack Platform environment. This custom configuration requires additional Heat templates included with your Overcloud stack. This section examines some of the custom configuration operations available to you.

10.1. 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, In this example, we aim to update the nameserver with a custom IP address on all nodes. We 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. We 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 our 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 only one Heat template. Subsequent usage overrides the Heat template to use.

10.2. CUSTOMIZING OVERCLOUD PRE-CONFIGURATION

The Overcloud uses Puppet for 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, we 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.

ExtraPreDeployments

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 our Heat template as the **`OS::TripleO::NodeExtraConfig`** resource type.

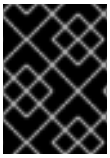
```
resource_registry:
  OS::TripleO::NodeExtraConfig: /home/stack/templates/nameserver.yaml

parameter_defaults:
  nameserver_ip: 192.168.1.1
```

To add 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 adds 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.

10.3. 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, we 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 our 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 add 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 adds 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.

10.4. CUSTOMIZING PUPPET CONFIGURATION DATA

There are two methods of passing Puppet configuration data to customize aspects of the Overcloud.

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.

10.5. 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 aim 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 our Puppet classes to install and configure **motd**.

Next, create an environment file (**/home/stack/templates/puppet_post_config.yaml**) that registers our 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.

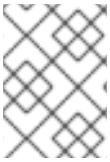
10.6. USING CUSTOMIZED OVERCLOUD HEAT TEMPLATES

When creating the Overcloud, the director uses a default set of Heat templates. However, it is possible to copy the standard Heat templates into a local directory and use these templates for creating your Overcloud. This is useful for customizing specific parts of your Overcloud.

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

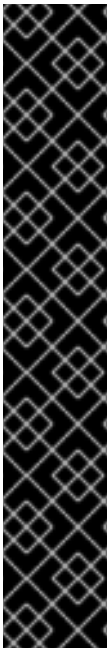
```
$ cp -r /usr/share/openstack-tripleo-heat-templates ~/templates/my-
overcloud
```

This creates a clone of the Overcloud Heat templates. When running **openstack overcloud deploy**, we use the **--templates** option to specify our local template directory. This occurs later in this scenario (see Section 6.2.9, “Creating the Advanced 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 10.1, “Customizing Configuration on First Boot”
- Section 10.2, “Customizing Overcloud Pre-Configuration”
- Section 10.3, “Customizing Overcloud Post-Configuration”
- Section 10.4, “Customizing Puppet Configuration Data”

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

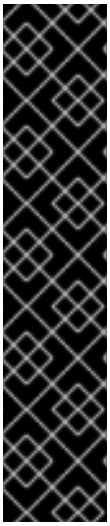
CHAPTER 11. UPDATING THE ENVIRONMENT

This chapter explores how to update your environment after creating your Overcloud of choice. This includes updating aspects of both the Undercloud and Overcloud.

11.1. UPDATING DIRECTOR PACKAGES

The director relies on standard RPM methods to update your environment. This involves ensuring your director's host uses the latest packages through **yum**:

```
$ sudo yum update
```



IMPORTANT

After the package update, make sure all OpenStack services are running properly on the director. Check the **ironic-api** and **ironic-discoverd** services are running. If not, please start them:

```
$ sudo systemctl restart openstack-ironic-api openstack-ironic-
discoverd
```

Likewise, **heat-engine** on the Undercloud can fail to start if its database is unavailable. If this occurs, restart **heat-engine** manually after the update:

```
$ sudo systemctl start openstack-heat-engine.service
```

11.2. UPDATING OVERCLOUD AND DISCOVERY IMAGES

This procedure ensures you have the latest images for node discovery and Overcloud deployment. Obtain these new images from the Red Hat Enterprise Linux OpenStack Platform downloads page on the Red Hat Customer Portal at https://access.redhat.com/downloads/content/191/ver=7.0/rhel---7/7.0/x86_64/product-downloads. See Section 3.7, “Obtaining Images for Overcloud Nodes” for more information on obtaining and extracting the image archives.

Download these images to the **images** directory on the **stack** user's home (**/home/stack/images**). After obtaining these images, follow this procedure to replace the images:

Procedure 11.1. Updating Images

1. Remove the existing images from the director.

```
$ openstack image list
$ openstack image delete [IMAGE-UUID] [IMAGE-UUID] [IMAGE-UUID]
[IMAGE-UUID] [IMAGE-UUID]
```

2. Import the latest images into the director.

```
$ cd ~/images
$ openstack overcloud image upload --update-existing
$ openstack baremetal configure boot
```

The director is now updated and using the latest packages and images. You do not need to restart any services after the update.

11.3. UPDATING THE OVERCLOUD

This section details the steps requires to update the Overcloud. Make sure to follow each section in order and only apply the sections relevant to your environment.

11.3.1. Configuration Agent

IMPORTANT

This section is only required if updating from Red Hat Enterprise Linux OpenStack Platform 7.0 or 7.1 to Red Hat Enterprise Linux OpenStack Platform 7.2 and later.

Due to a known issue (see BZ#1278181), the Overcloud requires some manual configuration of its configuration agent. This involves copying a new version of the configuration agent script from the director to each node in the Overcloud.

Log in as the **stack** user on the director host and source the Undercloud configuration:

```
$ source ~/stackrc
```

Copy the configuration agent (**55-heat-config**) to each Overcloud node. Use the following command to do this for all hosts:

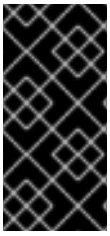
```
$ for i in `nova list|awk '/Running/ {print $(NF-1)}'|awk -F"=" '{print
$NF}'`; do echo $i; scp -o StrictHostKeyChecking=no /usr/share/openstack-
heat-templates/software-config/elements/heat-config/os-refresh-
config/configure.d/55-heat-config heat-admin@${i}: ; ssh -o
StrictHostKeyChecking=no heat-admin@${i} 'sudo /bin/bash -c "cp
/home/heat-admin/55-heat-config /usr/libexec/os-refresh-
config/configure.d/55-heat-config"'; done
```

This ensures the configuration agent is up-to-date.

This Overcloud also needs to recreate some post-deployment files. The director includes a script to achieve this. Copy and execute the **heat-config-rebuild-deployed** script on each node. Use the following command to do this for all nodes:

```
$ for i in `nova list|awk '/Running/ {print $(NF-1)}'|awk -F"=" '{print
$NF}'`; do echo $i; scp  -o StrictHostKeyChecking=no /usr/share/openstack-
heat-templates/software-config/elements/heat-config/bin/heat-config-
rebuild-deployed heat-admin@${i}: ; ssh -o StrictHostKeyChecking=no heat-
admin@${i} 'sudo /bin/bash -c "mkdir -p /usr/share/openstack-heat-
templates/software-config/elements/heat-config/bin ; cp heat-config-
rebuild-deployed /usr/share/openstack-heat-templates/software-
config/elements/heat-config/bin/heat-config-rebuild-deployed ; chmod +x
/usr/share/openstack-heat-templates/software-config/elements/heat-
config/bin/heat-config-rebuild-deployed ; /usr/share/openstack-heat-
templates/software-config/elements/heat-config/bin/heat-config-rebuild-
deployed"' ; done
```

11.3.2. Modified Overcloud Templates



IMPORTANT

This section is only required if using a modified template collection from Section 10.6, "Using Customized Overcloud Heat Templates". This is because the copy is a static snapshot of the original Heat template collection from **`/usr/share/openstack-tripleo-heat-templates/`**.

To update your modified template collection, you need to:

1. Backup your existing custom template collection:

```
$ mv ~/templates/my-overcloud/ ~/templates/my-overcloud.bak
```

2. Replace the new version of the template collection from **`/usr/share/openstack-tripleo-heat-templates`**:

```
# sudo cp -rv /usr/share/openstack-tripleo-heat-templates
~/templates/my-overcloud/
```

3. Check for differences between the old and new custom template collection. To see changes between the two, use the following **`diff`** command:

```
# diff -Nary ~/templates/my-overcloud.bak/ ~/templates/my-overcloud/
```

This helps identify customizations from the old template collection that you can incorporate into the new template collection.

4. Incorporate customization into the new custom template collection.

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 10.1, "Customizing Configuration on First Boot"
- Section 10.2, "Customizing Overcloud Pre-Configuration"
- Section 10.3, "Customizing Overcloud Post-Configuration"
- Section 10.4, "Customizing Puppet Configuration Data"

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

11.3.3. New Environment Parameters

The updated Heat template collection requires some new parameters not included in the original Red Hat Enterprise Linux OpenStack Platform 7.0 release. It is recommended to include these parameters in future updates.

Include the following additional parameters in a custom environment file (**`~/templates/param-updates.yaml`**):

Table 11.1. Additional Parameters to Include

New Parameter	Description
ControlPlaneDefaultRoute	The default route of the control plane network.
EC2MetadataIp	The IP address of the EC2 metadata server.

For example:

```
parameter_defaults:
  ControlPlaneDefaultRoute: 192.168.1.1
  EC2MetadataIp: 169.254.169.254
```

Make sure to include this file on any subsequent updates of the Overcloud.

11.3.4. Version Specific Notes

This section contains some notes specific to your initial version of the director and the Overcloud. Use this section as a reference for any inclusions to the Overcloud stack in subsequent updates.

If you started with OpenStack Platform director 7.0 and are upgrading to OpenStack Platform director 7.2 or later:

- Be sure to provide the same value for the **`ServiceNetMap`** parameter that was used on the initial cloud deployment (see Section 6.2.6.3, “Assigning OpenStack Services to Isolated Networks”. If a custom value was used on the initial deployment, provide the same custom value. If you are updating from 7.0 and used no custom **`ServiceNetMap`** value on the initial deployment, include the following environment file in the update command to preserve the 7.0 value:

  `/usr/share/openstack-tripleo-heat-templates/environments/updates/update-from-keystone-admin-internal-api.yaml`

  Make sure to include this file on any subsequent updates of the Overcloud.

  Changing the value of **`ServiceNetMap`** after Overcloud creation is not currently supported.

- If using a single network for the Overcloud (for example, the original deployment did not include **`network-isolation.yaml`**) then include the following environment file in the update command:

  `/usr/share/openstack-tripleo-heat-templates/environments/updates/update-from-publicvip-on-ctlplane.yaml`

Make sure to include this file on any subsequent updates of the Overcloud. Note that you do not need this file if using an external load balancer.

If you started with OpenStack Platform director 7.1 and are upgrading to OpenStack Platform director 7.2 or later:

- Be sure to provide the same value for the **`ServiceNetMap`** parameter that was used on the initial cloud deployment (see Section 6.2.6.3, "Assigning OpenStack Services to Isolated Networks". If a custom value was used on the initial deployment, provide the same custom value. If you are updating from 7.1 and used no custom value for **`ServiceNetMap`** on the initial deployment, then no additional environment file or value needs to be provided for **`ServiceNetMap`**. Changing the value of **`ServiceNetMap`** after Overcloud creation is not currently supported.

- Include the following environment file in the update command to make sure the VIP resources remain mapped to **`vip.yaml`**:

  `/usr/share/openstack-tripleo-heat-templates/environments/updates/update-from-vip.yaml`

  Make sure to include this file on any subsequent updates of the Overcloud. Note that you do not need this file if using an external load balancer.

- If updating from 7.1 and not using external load balancer, provide the control VIP for the **`control_virtual_ip`** input parameter. This is because the resource is replaced during the upgrade. To do so, find the current control_virtual_ip address with:

  ```
  $ neutron port-show control_virtual_ip | grep ip_address
  {"subnet_id": "3d7c11e0-53d9-4a54-a9d7-55865fcc1e47", "ip_address":
  "192.0.2.21"} |
  ```

  Add it into a custom environment file, such as **`~/templates/param-updates.yaml`** from Section 11.3.3, "New Environment Parameters", as follows:

  ```
  parameters:
    ControlFixedIPs: [{'ip_address':'192.0.2.21'}]
  ```

  Make sure to include this file on any subsequent updates of the Overcloud. Note that you do not need this file if using an external load balancer.

  Delete the existing Neutron port:

  ```
  $ neutron port-delete control_virtual_ip
  ```

  The update process replaces this VIP with a new port using the original IP address.

If upgrading from OpenStack Platform director 7.2 to OpenStack Platform director 7.3 or later:

- Heat on the Undercloud requires an increase in the RPC response timeout to accomodate a known issue (see BZ#1305947). Edit the **`/etc/heat/heat.conf`** and set the following parameter:

  ```
  rpc_response_timeout=600
  ```

Then restart all Heat services:

```
$ systemctl restart openstack-heat-api.service
$ systemctl restart openstack-heat-api-cfn.service
$ systemctl restart openstack-heat-engine.service
```

11.3.5. Updating the Overcloud Packages

The Overcloud relies on standard RPM methods to update the environment. This involves performing an update on all nodes using the **`openstack overcloud update`** from the director.

Use the **`-e`** to include environment files relevant to your Overcloud and its upgrade path. The order of the environment files is important as the parameters and resources defined in subsequent environment files take precedence. Use the following list as an example of the environment file order:

- The **`overcloud-resource-registry-puppet.yaml`** file from the Heat template collection. Although this file is included automatically when you run the **`openstack overcloud deploy`** command, **you must include this file when you run the `openstack overcloud update` command**.
- 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 6.2.6, “Isolating all Networks into VLANs” for more information on network islocation.
- Any external load balancing environment files.
- Any storage environment files.
- Any environment files for Red Hat CDN or Satellite registration.
- Any version-specific environment files from Section 11.3.4, “Version Specific Notes”.
- Any other custom environment files.

Running an update on all nodes in parallel might cause problems. For example, an update of a package might involve restarting a service, which can disrupt other nodes. This is why the update process updates each node using a set of breakpoints. This means nodes are updated one by one. When one node completes the package update, the update process moves on to the next node. The update process also requires the **`-i`** option, which puts the command in an interactive mode that requires confirmation at each breakpoint. Without the **`-i`** option, the update remains paused at the first breakpoint.

The following is an example update command for updating from director 7.1 to 7.2:

```
$ openstack overcloud update stack overcloud -i \
  --templates  \
  -e /usr/share/openstack-tripleo-heat-templates/overcloud-resource-
registry-puppet.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/network-
isolation.yaml \
  -e /home/stack/templates/network-environment.yaml \
  -e /home/stack/templates/storage-environment.yaml \
  -e /home/stack/templates/rhel-registration/environment-rhel-
registration.yaml \
```

```
  -e /usr/share/openstack-tripleo-heat-
templates/environments/updates/update-from-vip.yaml \
  -e /home/stack/templates/param-updates.yaml
```

Running this command starts the update process. During this process, the director will periodically prompt you to clear breakpoints. For example:

```
not_started: [u'overcloud-controller-0', u'overcloud-controller-1',
u'overcloud-controller-2']
on_breakpoint: [u'overcloud-compute-0']
Breakpoint reached, continue?
```

Press Enter to clear the breakpoint from last node on the **on_breakpoint** list. This begins the update for that node. You can also type a node name to clear a breakpoint on a specific node, or a regular expression to clear breakpoints on mulitple nodes at once. However, it is not recommended to clear breakpoints on multiple controller nodes at once. Continue this process until all nodes have complete their update.

IMPORTANT

The update process does not reboot any nodes in the Overcloud automatically. See Chapter 9, *Rebooting the Overcloud* for steps on how to reboot your environment without instance downtime.

IMPORTANT

If you configured fencing for your Controller nodes, the update process might disable it. When the update process completes, reenable fencing with the following command on one of the Controller nodes:

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

CHAPTER 12. 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-discoverd** uses two units as well: **openstack-ironic-discoverd** and **openstack-ironic-discoverd-dnsmasq**. Use both units for each respective component. For example:

```
$ sudo journalctl -u openstack-ironic-discoverd -u openstack-ironic-discoverd-dnsmasq
```

- **ironic-discoverd** also stores the ramdisk logs in **/var/log/ironic-discoverd/ramdisk/** as gz-compressed tar files. Filenames contain date, time, and IPMI address of the node. Use these logs for diagnosing introspection issues.

12.1. 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:

Procedure 12.1. Fixing an Incorrect MAC Address

1. Find out the assigned port UUID:

```
$ ironic node-port-list [NODE UUID]
```

2. Update the MAC address:

```
$ ironic port-update [PORT UUID] replace address=[NEW MAC]
```

Procedure 12.2. Fix an Incorrect IPMI Address

- Run the following command:

```
$ ironic node-update [NODE UUID] replace driver_info/ipmi_address=[NEW IPMI ADDRESS]
```

12.2. TROUBLESHOOTING HARDWARE INTROSPECTION

The discovery and introspection process must run to completion. However, Ironic's Discovery Daemon (**ironic-discoverd**) 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 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-discoverd**, 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
```

Stopping the Discovery Process

Currently **ironic-discoverd** 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-discoverd/discoverd.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:

Procedure 12.3. Stopping the Discovery Process

1. Change the power state of each node to off:

   ```
   $ ironic node-set-power-state [NODE UUID] off
   ```

2. Remove **ironic-discoverd** cache and restart it:

   ```
   $ rm /var/lib/ironic-discoverd/discoverd.sqlite
   $ sudo systemctl restart openstack-ironic-discoverd
   ```

12.3. 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.

12.3.1. Orchestration

In most cases, Heat shows the failed overcloud stack after 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 indicate an issue with the initial orchestration setup. Check your Heat templates and configuration options, and check for any error messages after running **`openstack overcloud deploy`**.

12.3.2. 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.

- Check the **`Provision State`** and **`Maintenance`** columns in the resulting table. Check for the following:
  - An empty table or less nodes that you expect
  - **`Maintenance`** is set to True
  - **`Provision State`** is set to **`manageable`**

  This usually indicates an issue from the registration or discovery processes. For example, if **`Maintenance`** sets 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 the 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 finished for this node yet. 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.

12.3.3. 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.

Procedure 12.4. Diagnosing Post-Deployment Configuration Issues

1. 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**.

2. 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.

3. Use the **nova** command to see the IP addresses of the Overcloud nodes.

```
$ nova list
```

   Login 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, login to a Controller node. The **heat-admin** user has sudo access.

```
$ ssh heat-admin@192.0.2.14
```

4. Check the **os-collect-config** log for a possible reason for the failure.

```
$ sudo journalctl -u os-collect-config
```

5. 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 12.5, “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/***

6. 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
```

12.4. AVOID 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:

Procedure 12.5. Identify active IP addresses

1. Install **nmap**:

```
# yum install nmap
```

2. 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
```

3. 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 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
```

12.5. 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 enough nodes are in **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
```

12.6. TROUBLESHOOTING THE OVERCLOUD AFTER CREATION

After creating your Overcloud, you might aim 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.

12.6.1. 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, then applies 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 12.3, “Troubleshooting Overcloud Creation”. 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.

12.6.2. Controller Service Failures

The Overcloud Controller nodes contain the bulk of Red Hat Enterprise Linux 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 on 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 in **/var/log/**.

12.6.3. Compute Service Failures

Compute nodes use the OpenStack Nova 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 virtual machines from the host to an operational Compute node, then disable the node. See Section 7.8, “Migrating VMs from an Overcloud Compute Node” for more information on node migrations.

12.6.4. Ceph Storage Service Failures

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

12.7. TUNING THE UNDERCLOUD

The advice in this section aims to help increase the performance of your Undercloud. Implement the recommendations as necessary.

- The OpenStack Authentication service (**keystone**) uses a token-based system for access to other OpenStack services. After a certain period, the database accumulates many unused tokens. It is recommended to create a cronjob to flush the token table in the database. For example, to flush the token table at 4 a.m. each day:

```
0 04 * * * /bin/keystone-manage token_flush
```

- 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
```

12.8. IMPORTANT LOGS FOR UNDERCLOUD AND OVERCLOUD

Use the following logs to find out information about the Undercloud and Overcloud when troubleshooting.

Table 12.1. Important Logs for Undercloud and Overcloud

Information	Undercloud or Overcloud	Log Location

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-discoverd/***
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-*/***

Information	Undercloud or Overcloud	Log Location
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. COMPONENTS

This section contains a list of components that the director uses.

Shared Libraries

diskimage-builder

`diskimage-builder` is an image building tool.

dib-utils

`dib-utils` contains tools that `diskimage-builder` uses.

os-collect-config, os-refresh-config, os-apply-config, os-net-config

A suite of tools used to configure instances.

tripleo-image-elements

`tripleo-image-elements` is a repository of `diskimage-builder` style elements for installing various software components.

Installer

instack

`instack` executes `diskimage-builder` style elements on the current system. This enables a current running system to have an element applied in the same way that `diskimage-builder` applies the element to an image build.

instack-undercloud

`instack-undercloud` is the Undercloud installer based around `instack`.

Node Management

ironic

The OpenStack Ironic project is responsible for provisioning and managing bare metal instances.

ironic-discoverd

`ironic-discoverd` discovers hardware properties for newly enrolled nodes.

Deployment Planning

tuskar

The OpenStack Tuskar project is responsible for planning of deployments

Deployment and Orchestration

heat

The OpenStack Heat project is an orchestration tool. It reads YAML files describing the OpenStack environment's resources and sets those resources into a desired state.

heat-templates

The **`openstack-heat-templates`** repository contains additional image elements for producing disk images for Puppet configuration using Heat.

tripleo-heat-templates

The **`openstack-tripleo-heat-templates`** repository describe the OpenStack environment in Heat Orchestration Template YAML files and Puppet manifests. Tuskar processes these templates, which develop into an actual environment through Heat.

puppet-modules

OpenStack Puppet modules are used to configure the OpenStack environment through **`tripleo-heat-templates`**.

tripleo-puppet-elements

The **`tripleo-puppet-elements`** describe the contents of disk images which the director uses to install Red Hat Enterprise Linux OpenStack Platform.

User Interfaces

tuskar-ui

Provides a GUI to install and manage OpenStack. It is implemented as a plugin to the Horizon dashboard.

tuskar-ui-extras

Provides GUI enhancements for **`tuskar-ui`**. It is implemented as a plugin to the Horizon dashboard.

python-openstackclient

The **`python-openstackclient`** is a CLI tool that manages multiple openstack services and clients.

python-rdomanager-oscplugin

The **`python-rdomanager-oscplugin`** is a CLI tool embedded into **`python-openstackclient`**. It provides functions related to **`instack`** installation and initial configuration.

APPENDIX B. SSL/TLS CERTIFICATE CONFIGURATION

As an optional part of the processes outlined in Section 3.6, “Configuring the Director” or Section 6.2.7, “Enabling SSL/TLS on the Overcloud”, you can set the use SSL/TLS for communication on either the Undercloud or Overcloud. However, if using an SSL/TLS certificate with your own certificate authority, the certificate requires a certain configuration for use.

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 the a certificate file called **`ca.crt.pem`**. Copy this file to each client that aims to access your Red Hat Openstack Platform environment and run the following command 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
```

CREATING AN SSL/TLS CERTIFICATE

This next procedure creates a signed certificate for either the Undercloud and Overcloud.

Copy the default OpenSSL configuration file for customization.

```
$ cp /etc/pki/tls/openssl.cnf .
```

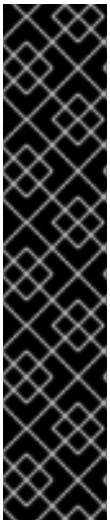
Edit the custom **`openssl.cnf`** file and set SSL parameters to use for the director. An example of the types of parameters to modify include:

```
[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 = 192.168.0.1
DNS.2 = instack.localdomain
DNS.3 = vip.localdomain
```



IMPORTANT

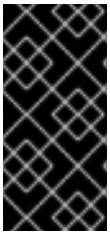
Set the **commonName_default** to the IP address of the Public API:

- For the Undercloud, use the **undercloud_public_vip** parameter in **undercloud.conf**.
- 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.

Include the same Public API IP address as an IP entry and a DNS entry in the **alt_names** section. If also using DNS, include the hostname for the server as DNS entries in the same section. For more information about **openssl.cnf**, run **man openssl.cnf**.

Run the following commands to generate the key (**server.key.pem**), the certificate signing request (**server.csr.pem**), and the signed certificate (**server.crt.pem**):

```
$ openssl genrsa -out server.key.pem 2048
$ openssl req -config openssl.cnf -key server.key.pem -new -out
server.csr.pem
$ sudo openssl ca -config openssl.cnf -extensions v3_req -days 3650 -in
server.csr.pem -out server.crt.pem -cert ca.cert.pem
```



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 this key pair a SSL/TLS certificate for either the Undercloud or Overcloud.

USING THE CERTIFICATE WITH THE UNDERCLOUD

Run the following command to create the certificate:

```
$ cat server.crt.pem server.key.pem > undercloud.pem
```

This creates a **undercloud.pem** for use with the **undercloud_service_certificate** option. 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 certificate authority to the Undercloud's list of trusted Certificate Authorities:

```
$ sudo cp ca.crt.pem /etc/pki/ca-trust/source/anchors/
$ sudo update-ca-trust extract
```

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
```

Continue installing the Undercloud as per the instructions in Section 3.6, “Configuring the Director”.

USING THE CERTIFICATE WITH THE OVERCLOUD

Use the certificate with the **enable-tls.yaml** file from Section 6.2.7, “Enabling SSL/TLS on the Overcloud”.

APPENDIX C. 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 either Section 6.1.1, “Registering Nodes for the Basic Overcloud” or Section 6.2.1, “Registering Nodes for the Advanced Overcloud”.

C.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.

C.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.

Additional Notes

- 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).

C.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"
```

Additional Notes

- 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
```

C.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**.

Additional Notes

- 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
```

C.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 Enterprise Linux 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.

Additional Notes

- 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.

C.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 Enterprise Linux OpenStack Platform enterprise environments.

pm_type

Set this option to **fake_pxe**.

Additional Notes

- 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.
- 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 D. AUTOMATED HEALTH CHECK (AHC) TOOLS PARAMETERS

The following tables present a reference for the different parameters you can use for AHC Tools policies.

D.1. HARD DRIVE

AHC Tools report disks's properties from:

1. Regular SATA controllers or logical drives from RAID controllers
2. Disks attached to a Hewlett Packard RAID Controller

Table D.1. Regular SATA Controller Parameters

Value	Description	Sample Configuration	Discrimination Level
size	Size of the disk	('disk', 'sda', 'size', '899')	Medium
vendor	Vendor of the disk	('disk', 'sda', 'vendor', 'HP')	Medium
model	Model of the disk	('disk', 'sda', 'model', 'LOGICAL VOLUME')	High
rev	Firmware revision of the disk	('disk', 'sda', 'rev', '3.42')	Medium
WCE	Write Cache Enabled	('disk', 'sda', 'WCE', '1')	Low
RCD	Read Cache Disabled	('disk', 'sda', 'RCD, '1')	Low

Table D.2. Hewlett Packard Raid Controller Parameters

Value	Description	Sample Configuration	Discrimination Level
size	Size of the raw disk	('disk', '1I:1:1', 'size', '300')	Medium
type	Type of the raw disk	('disk', '1I:1:1', 'type', 'SAS')	Low
slot	Raw disk slot's id	('disk', '1I:1:1', 'slot', '0')	Medium

D.2. SYSTEM

Product information is provided by the DMI structures of the host. This information is not always provided by the hardware manufacturer.

Table D.3. System Product Parameters

Value	Description	Sample Configuration	Discrimination Level
serial	Serial number of the hardware	('system', 'product', 'serial', 'XXXXXX')	Unique*
name	Product name	('system', 'product', 'name', 'ProLiant DL360p Gen8 (654081-B21)')	High
vendor	Vendor name	('system', 'product', 'vendor', 'HP')	Medium

Table D.4. System IPMI Parameters

Value	Description	Sample Configuration	Discrimination Level
ipmi	The IPMI channel number	('system', 'ipmi', 'channel', 2)	Low
ipmi-fake	Fake IPMI interface for testing	('system', 'ipmi-fake', 'channel', '0')	Low

D.3. FIRMWARE

Firmware information is provided by the DMI structures of the host. This information is not always provided by the hardware manufacturer.

Table D.5. Firmware Parameters

Value	Description	Sample Configuration	Discrimination Level
version	Version of the BIOS	('firmware', 'bios', 'version', 'G1ET73WW (2.09)')	Medium
date	Date of the BIOS release	('firmware', 'bios', 'date', '10/19/2012')	Medium
vendor	Vendor	('firmware', 'bios', 'vendor', 'LENOVO')	Low

D.4. NETWORK

Table D.6. Network Parameters

Value	Description	Sample Configuration	Discrimination Level
serial	MAC address	('network', 'eth0', 'serial', 'd8:9d:67:1b:07:e4')	Unique
vendor	NIC's vendor	('network', 'eth0', 'vendor', 'Broadcom Corporation')	Low
product	NIC's description	('network', 'eth0', 'product', 'NetXtreme BCM5719 Gigabit Ethernet PCIe')	Medium
size	Link capability in bits/sec	('network', 'eth0', 'size', '1000000000')	Low
ipv4	IPv4 address	('network', 'eth0', 'ipv4', '10.66.6.136')	High
ipv4-netmask	IPv4 netmask	('network', 'eth0', 'ipv4-netmask', '255.255.255.0')	Low
ipv4-cidr	IPv4 cidr	('network', 'eth0', 'ipv4-cidr', '24')	Low
ipv4-network	IPv4 network address	('network', 'eth0', 'ipv4-network', '10.66.6.0')	Medium
link	Physical Link Status	('network', 'eth0', 'link', 'yes')	Medium
driver	NIC's driver name	('network', 'eth0', 'driver', 'tg3')	Low
duplex	NIC's duplex type	('network', 'eth0', 'duplex', 'full')	Low
speed	NIC's current link speed	('network', 'eth0', 'speed', '10Mbit/s')	Medium
latency	PCI latency of the network device	('network', 'eth0', 'latency', '0')	Low
autonegotiation	NIC's auto-negotiation	('network', 'eth0', 'autonegotiation', 'on')	Low

D.5. CPU

Table D.7. Per CPU Parameters

Value	Description	Sample Configuration	Discrimination Level
physid	CPU's physical ID	('cpu', 'physical_0', 'physid', '1')	Low
cores	CPU's number of cores	('cpu', 'physical_0', 'cores', '2')	Medium
enabled_cores	CPU's number of enabled cores	('cpu', 'physical_0',' enabled_cores', '2')	Medium
threads	CPU's number of threads	('cpu', 'physical_0', 'threads', '4')	Medium
product	CPU's identification string	('cpu', 'physical_0', 'product', 'Intel(R) Core(TM) i5-3320M CPU @ 2.60GHz')	High
vendor	CPU's vendor	('cpu', 'physical_0', 'vendor', 'Intel Corp.')	Low
frequency	CPU's internal frequency in Hz	('cpu', 'physical_0', 'frequency', '1200000000')	Low
clock	CPU's clock in Hz	('cpu', 'physical_0', 'clock', '100000000')	Low

Table D.8. CPU Aggregate Parameters

Value	Description	Sample Configuration	Discrimination Level
number (physical)	Number of physical CPUs	('cpu', 'physical', 'number', 2)	Medium
number (logical)	Number of logical CPUs	('cpu', 'logical', 'number', '8')	Medium

D.6. MEMORY

Memory information is provided by the DMI structures of the host. This information is not always provided by the hardware manufacturer.

Table D.9. Memory Parameters

Value	Description	Sample Configuration	Discrimination Level

Value	Description	Sample Configuration	Discrimination Level
total	Amount of memory on the host in bytes	('memory', 'total', 'size', '17179869184')	High
size	Bank size in bytes	('memory', 'bank:0', 'size', '4294967296')	Medium
clock	Memory clock speed in Hz	('memory', 'bank:0', 'clock', '667000000')	Low
description	Memory description	('memory', 'bank:0', 'description', 'FB-DIMM DDR2 FB-DIMM Synchronous 667 MHz (1.5 ns)')	Medium
vendor	Memory vendor	('memory', 'bank:0', 'vendor', 'Nanya Technology')	Medium
serial	Memory serial number	('memory', 'bank:0', 'serial', 'C7590943')	Unique*
slot	Physical slot of this Bank	('memory', 'bank:0', 'slot', 'DIMM1')	High
banks	Number of memory banks	('memory', 'banks', 'count', 8)	Medium

D.7. INFINIBAND

Table D.10. Infiniband Per Card Parameters

Value	Description	Sample Configuration	Discrimination Level
card_type	Card's type	('infiniband', 'card0', 'card_type', 'mlx4_0')	Medium
device_type	Card's device type	('infiniband', 'card0', 'device_type', 'MT4099')	Medium
fw_version	Card firmware version	('infiniband', 'card0', 'fw_version', '2.11.500')	High
hw_version	Card's hardware version	('infiniband', 'card0', 'hw_version', '0')	Low
nb_ports	Number of ports	('infiniband', 'card0', 'nb_ports', '2')	Low

Value	Description	Sample Configuration	Discrimination Level
sys_guid	Global unique ID of the card	('infiniband', 'card0', 'sys_guid', '0x0002c90300ea7183')	Unique
node_guid	Global unique ID of the node	('infiniband', 'card0', 'node_guid', '0x0002c90300ea7180')	Unique

Table D.11. Infiniband Per Port Parameters

Value	Description	Sample Configuration	Discrimination Level
state	Interface state	('infiniband', 'card0_port1', 'state', 'Down')	High
physical_state	Physical state of the link	('infiniband', 'card0_port1', 'physical_state', 'Down')	High
rate	Speed in Gbit/sec	('infiniband', 'card0_port1', 'rate', '40')	High
base_lid	Base local ID of the port	('infiniband', 'card0_port1', 'base_lid', '0'	Low
lmc	Local ID mask count	('infiniband', 'card0_port1', 'lmc', '0')	Low
sm_lid	Subnet manager local ID	('infiniband', 'card0_port1', 'sm_lid', '0')	Low
port_guid	Global unique ID of the port	('infiniband', 'card0_port1', 'port_guid', '0x0002c90300ea7181')	Unique

APPENDIX E. NETWORK INTERFACE PARAMETERS

The following table defines the Heat template parameters for network interface types.

Table 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

Table 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

Option	Default	Description
persist_mapping	False	Write the device alias configuration instead of the system names

Table 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

Table 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

Option	Default	Description
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

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 i.e. 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.

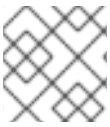
F.2. CONFIGURING ROUTES AND DEFAULT ROUTES

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, we 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, we 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 Neutron's external bridge mapping. This maps the physical interface to the **br-int** instead of the 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

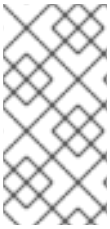
When moving the address (and possibly route) statements onto the bridge, remove the corresponding VLAN interface from the bridge. Make the changes to all applicable roles. The External network is only on the controllers, so only the controller template requires a change. The Storage network on the other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

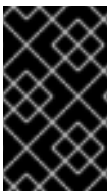
Parameter	Description	Example
InternalApiNetCidr	The network and subnet for the Internal API network	172.17.0.0/24
StorageNetCidr	The network and subnet for the Storage network	
StorageMgmtNetCidr	The network and subnet for the Storage Management network	
TenantNetCidr	The network and subnet for the 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	
ExternalNetworkVlanID	The VLAN ID for the External network	

Parameter	Description	Example
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	The network and subnet for the Provisioning network	ControlPlaneSubnetCidr: 192.0.2.0/24
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"]
NeutronExternalNetworkBridge	Defines the bridge to use for the External network. Set to **`"br-ex"`** if using floating IPs on native VLAN on bridge **`br-ex`**.	NeutronExternalNetworkBridge: "br-ex"
BondInterfaceOvsOptions	The options for bonding interfaces	bond_mode=balance-tcp lacp=active other-config:lacp-fallback-ab=true"

APPENDIX H. BONDING OPTIONS

The Overcloud provides networking through Open vSwtich, which provides several options for bonded interfaces. In Section 6.2.6.2, “Creating an Advanced Overcloud Network Environment File”, we configure a bonded interface in the network environment file using the following option:

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



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.

The following table provides some explanation of these options and some alternatives depending on your hardware. In addition, you can still use LACP with Linux bonding.

Table H.1. Bonding Options

bond_mode=balance-tcp	This mode will perform load balancing by taking layer 2 to layer 4 data into consideration. For example, destination MAC address, IP address, and TCP port. In addition, **balance-tcp** requires that LACP be configured on the switch. This mode is similar to mode 4 bonds used by the Linux bonding driver. **balance-tcp** is recommended when possible, as LACP provides the highest resiliency for link failure detection, and supplies additional diagnostic information about the bond. The recommended option is to configure **balance-tcp** with LACP. This setting attempts to configure LACP, but will fallback to **active-backup** if LACP cannot be negotiated with the physical switch.
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**.
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 slave/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 packet-loop via the standby interface.

APPENDIX I. DEPLOYMENT PARAMETERS

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

Table I.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
-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]	Defines the flat networks to configure in neutron plugins. Defaults to "datacentre" to permit external network creation	datacentre

Parameter	Description	Example
--neutron-physical-bridge [NEUTRON_PHYSICAL_BRIDGE]	An Open vSwitch bridge to create on each hypervisor. This defaults to "br-ex". Typically, this should not need to be changed	br-ex
--neutron-bridge-mappings [NEUTRON_BRIDGE_MAPPINGS]	The logical to physical bridge mappings to use. Defaults to mapping the external bridge on hosts (br-ex) to a physical name (datacentre). We use this for the default floating network	datacentre:br-ex
--neutron-public-interface [NEUTRON_PUBLIC_INTERFACE]	Defines the interface to bridge onto br-ex for network nodes	nic1, eth0
--hypervisor-neutron-public-interface [HYPERVISOR_NEUTRON_PUBLIC_INTERFACE]	What interface to add to the bridge on each hypervisor	nic1, eth0
--neutron-network-type [NEUTRON_NETWORK_TYPE]	The tenant network type for Neutron	gre or vxlan
--neutron-tunnel-types [NEUTRON_TUNNEL_TYPES]	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]	Ranges of GRE tunnel IDs to make available for tenant network allocation	1:1000
--neutron-vni-ranges [NEUTRON_VNI_RANGES]	Ranges of VXLAN VNI IDs to make available for tenant network allocation	1:1000
--neutron-disable-tunneling	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]	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]	The mechanism drivers for the Neutron tenant network. Defaults to "openvswitch". To specify multiple values, use a comma separated string	'openvswitch,l2population'

Parameter	Description	Example
--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
--cinder-lvm	Use the LVM iSCSI driver for Cinder storage	
--tripleo-root [TRIPLEO_ROOT]	The directory for the director's configuration files. Leave this as the default	
--nodes-json [NODES_JSON]	The original JSON file used for node registration. The director provides some modifications to this file after creating your Overcloud. Defaults to instackenv.json	
--no-proxy [NO_PROXY]	Defines custom values for the environment variable no_proxy, which excludes certain domain extensions from proxy communication	
-O [OUTPUT DIR], --output-dir [OUTPUT DIR]	Directory to write Tuskar template files into. It will be created if it does not exist. If not provided a temporary directory will be used	~/templates/plan-templates
-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

Parameter	Description	Example
--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.	
--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 6 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	

APPENDIX J. REVISION HISTORY

Revision 7.3-18 **Thu Jun 15 2017** **Dan Macpherson**
Adding note regarding multiple bonds on a bridge

Revision 7.3-17 **Thu Mar 30 2017** **Dan Macpherson**
Emphasis added to stack update command

Revision 7.3-16 **Wed Sep 21 2016** **Dan Macpherson**
Fixing another OSD mapping

Revision 7.3-15 **Wed Sep 21 2016** **Dan Macpherson**
Correction to OSD layout

Revision 7.3-14 **Mon Aug 22 2016** **Dan Macpherson**
Adding note about disk space for Undercloud

Revision 7.3-13 **Tue Aug 16 2016** **Dan Macpherson**
Revising fencing section for clarity

Revision 7.3-12 **Thu Aug 4 2016** **Dan Macpherson**
Backporting Scaling Chapter from OSP 8 Director Installation and Usage Guide

Revision 7.3-11 **Thu Jun 16 2016** **Dan Macpherson**
Adding note for SAN support

Revision 7.3-10 **Fri May 27 2016** **Dan Macpherson**
Adding note about UEFI boot mode support for nodes

Revision 7.3-9 **Tue Apr 26 2016** **Dan Macpherson**
Minor updates

Revision 7.3-8 **Fri Apr 8 2016** **Dan Macpherson**
Adding Removal Protection

Revision 7.3-7 **Fri Apr 8 2016** **Dan Macpherson**
Missing sudo access for some Ceph commands

Revision 7.3-6 **Tue Apr 5 2016** **Dan Macpherson**
Test build

Revision 7.3-5 **Tue Apr 5 2016** **Dan Macpherson**
Added Ceph Storage replacement instructions

Revision 7.3-4 **Thu Mar 3 2016** **Dan Macpherson**
Adding Repository Requirements for Satellite

Revision 7.3-3 **Wed Mar 2 2016** **Dan Macpherson**
Minor change to DNS listing for SSL

Revision 7.3-2 **Tue Mar 1 2016** **Dan Macpherson**
Restructuring node scaling content

Revision 7.3-1 **Thu Feb 18 2016** **Dan Macpherson**

Adding new content for OpenStack Platform director 7.2
Added SSL/TLS for Overcloud instructions
Added Satellite 5 registration for Overcloud information
Added log list
Various minor fixes

Revision 7.2-1 **Sun Dec 20 2015** **Dan Macpherson**
Adding new content for OpenStack Platform director 7.2, including new update procedure, registration information, and various minor additions.

Revision 7.1-14 **Wed Dec 16 2015** **Dan Macpherson**
Adding NFS option for SELinux context

Revision 7.1-13 **Tue Dec 15 2015** **Dan Macpherson**
Minor changes to custom Puppet config section

Revision 7.1-12 **Fri Dec 11 2015** **Dan Macpherson**
Correction to nova migration commands

Revision 7.1-11 **Fri Dec 11 2015** **Dan Macpherson**
Adding admonition for LACP issue

Revision 7.1-10 **Tue Dec 8 2015** **Dan Macpherson**
Adding information about deprecation of LACP on OVS

Revision 7.1-9 **Wed Dec 2 2015** **Dan Macpherson**
Corrections to custom preconfiguration content

Revision 7.1-7 **Tue Dec 1 2015** **Dan Macpherson**
Adding ExtraConfig resources and hiera data params

Revision 7.1-6 **Mon Nov 30 2015** **Dan Macpherson**
Adding fixes for replacing HA Controller node routine

Revision 7.1-5 **Thu Nov 19 2015** **Dan Macpherson**
Updating documentation for OSPd 7y2

Revision 7.1-4 **Wed Oct 14 2015** **Dan Macpherson**
Adding stack update instructions
Adding UCS Fencing link

Revision 7.1-2 **Fri Oct 9 2015** **Martin Lopes**
Added note about entitlement consumption during deployments

Revision 7.1-1 **Fri Oct 9 2015** **Dan Macpherson**
Adding stack update instructions

Revision 7.1-2 **Fri Oct 9 2015** **Dan Macpherson**
Changes to the upgrade process

Revision 7.1-1 **Fri Oct 9 2015** **Dan Macpherson**
Fixing download link

Revision 7.1-0 **Thu Oct 8 2015** **Dan Macpherson**
Static IP for Provisioning network
New validation methods

Revision 7.0-18 **Wed Oct 7 2015** **Dan Macpherson**
Adding clarification for --reg-sat-url parameter

Revision 7.0-17 **Tue Oct 6 2015** **Dan Macpherson**
Adding clarifications for networking and image uploading

Revision 7.0-16 **Tues Oct 6 2015** **Martin Lopes**
Added troubleshooting advice for duplicate IP addresses in the Provisioning network.

Revision 7.0-15 **Fri Oct 2 2015** **Dan Macpherson**
Minor corrections

Revision 7.0-14 **Thu Oct 1 2015** **Dan Macpherson**
Added sudo access for AHC Tools

Revision 7.0-13 **Mon Sep 28 2015** **Dan Macpherson**
Adding extra fencing devices

Revision 7.0-12 **Fri Sep 25 2015** **Dan Macpherson**
Corrected Admin and Public IP settings
Added yum-plugin-priorities documentation to avoid repository and package conflicts
Various minor fixes and revisions

Revision 7.0-11 **Thu Sep 24 2015** **Dan Macpherson**
Added new instructions for registering and configuring new nodes for scaling
Revised External network creation in post-deployment section
Minor corrections

Revision 7.0-10 **Fri Sep 18 2015** **Dan Macpherson**
Adding commands for single node introspection and registering additional nodes

Revision 7.0-9 **Fri Sep 11 2015** **Dan Macpherson**
Adding troubleshooting advice for Overcloud post-creation

Revision 7.0-8 **Tue Sep 8 2015** **Dan Macpherson**
Fixing pxe_ssh instructions
Correcting NeutronExternalNetworkBridge syntax for Basic and Advanced scenarios
Adding missing baremetal flavor creation
Providing correct SELinux context for SSL certificate
Adding introspection progress command
Adding Ceph customization requirements

Revision 7.0-7 **Mon Aug 24 2015** **Dan Macpherson**
Adding fixes to GUI Test Scenario
Removing Tuskar-based content (except for Test Scenario)

Revision 7.0-6 **Mon Aug 17 2015** **Dan Macpherson**
Added instructions on unpacking image archive
Minor corrections

Revision 7.0-5 **Mon Aug 10 2015** **Dan Macpherson**
Adding additional storage environment fix
Minor corrections

Revision 7.0-4 **Thu Aug 6 2015** **Dan Macpherson**
Modifying text for -i option when updating Overcloud packages

Revision 7.0-3 **Thu Aug 6 2015** **Dan Macpherson**

Updating warning about environment files

Correcting SELinux rule for SSL

Updating Overcloud image links

Revision 7.0-2 **Wed Aug 5 2015** **Dan Macpherson**

Building with new sort order

Revision 7.0-1 **Wed May 20 2015** **Dan Macpherson**

Initial draft of documentation